UNIVERSITY OF OSLO
Department of Informatics

Metamodel-based Editor for Service Oriented Architecture (MED4SOA)

Master thesis

Rudin Gjataj

July 5, 2007

. . . to my grandparents, R. and A. Rama,
whose love and devotion brought me so far . . .

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Arne-Jørgen Berre for his guidance in this master thesis. His approach to mentoring, allowed me to make an independent research project, but his support, contribution and encouragement was always appreciated, in good or bad moments.

Many thanks will go to SINTEF and all the employees of the Department of Information and Communication Technology, Cooperative and Trusted Systems. Especially, to Brian Elvesæter, which contribution in the end phase was much appreciated. Thank you all for making me a part of the team and for giving me the opportunity to work with the ATHENA research project.

Special thanks and appreciations goes to Stig Torsbakken and Visar Isaj, for reading through this thesis and providing me with the most needed commentary and corrections.

In the end, to all the people that were beside me in this long journey: family, friends, colleagues, thank you for everything. Without your inspiration and encouragement I would not be here, writing these last words.

Oslo, July 2007

Rudin Gjataj

Abstract

In software development tool support is essential. Since the standardization of UML and Model Driven Architecture (MDA), new approaches in the design and implementation of software systems have flourished. These approaches are specific architectures like Service Oriented Architecture (SOA) or specialised MDA flavours, like Model Driven Software Development (MDSD).

In this thesis we provide a Metamodel-based Editor for Service Oriented Architecture (MED4SOA). It is a graphical modeling editor, which lays the fundamentals of designing SOA-based systems using a Domain Specific Language (DSL). MED4SOA constitutes of two main parts: the metamodel capturing the concepts of the service domain and the graphical editing framework which realizes the visualisation of the metamodel concepts, resulting in a graphical editor. The metamodel and graphical editing framework were chosen after an investigation of the state of the art for these technologies. MED4SOA is used to build models, which contain the concepts defined in the Platform Independent Model for Service Oriented Architecture (PIM4SOA), while it is constructed, generated and run using Eclipse Graphical Modeling Framework (GMF). We explain in detail the concepts captured in the metamodel and the architecture of the editor, including the underlaying technologies of GMF, respectively Eclipse Modeling Framework (EMF) and Graphical Eclipse Framework (GEF).

MED4SOA is a collection of ideas and technologies put together to support the emerging of SOA, using principles of MDA, MDSD and Language Driven Development (LDD). Even though in this thesis we introduce a prototype of MED4SOA, its development and our experience in the process provides a precedent in the development of a graphical framework for SOA. From our experience we point out the potential and drawbacks of both PIM4SOA and GMF.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"'Where shall I begin, please your Majesty?' he asked. 'Begin at the beginning,' the King said, gravely, 'and go on till you come to the end: then stop."

Lewis Carroll

SINCE 1997, when Unified Modeling Language (UML) was standardized and adopted by the Object Management Group (OMG), modeling has become a significant part of software engineering [1]. It spreads in different cycles of the development process and requires tool support in the construction of models. The tools support has been addressed with the introduction of different commercial and open source tools based on UML. These tools give the developers the opportunity to create models, which are further used as documentation, or in model transformations and code generation.

But the world has evolved since then. Nowadays, the tool support to software engineering is essential. It is not a commodity, but a necessity to achieve a higher pace in the development process [2]. The introduction of UML have surely made the application of models more popular in software development, but also opened the way for new ideas. An alternative to UML is a Domain Specific Language (DSL), a language tailored for a specific domain. Similarly to UML, a DSL needs a modeling tool to support the language.

The popularity of models in software engineering has increased through the standardization of Model Driven Architecture (MDA). This architecture provides an approach to business challenge and technology change [3]. Another architecture focus on the relation of business to technology is SOA. Being a conceptual framework, SOA allows business functionality or application logic to be available through IT services [4].

In this thesis we present a Metamodel-based Editor for Service Oriented Architecture (MED4SOA). It is a graphical modeling editor built to support a DSL for SOA. The editor builds on a given metamodel, Platform Independent Model for Service Oriented Architecture (PIM4SOA), which captures the concepts and abstractions of the service domain. These concepts and abstractions are realized graphically in Graphical Modeling Framework (GMF). The framework generates the editor that provides facilities in the construction of models describing a SOA-based system.

1.1 Motivation

In the present, there is no standard modeling language tailored for SOA. Developers design their systems using UML diagrams, which are further used as documentation or to generate skeleton code. Based on MDA [3], models should be a central artefact in the development process. They are the basics to transformation and code generation. In this context we are motivated by the need of a DSL for services, which describe the system in a platform independent way, and an editor that supports this language.

Another motivation is the dilemma of software architects in choosing the implementation platform for a SOA- based system. Web Services [5] are a wide spread implementation for an SOA, but in the last years new technologies have emerged. Without a platform independent DSL and its editor, the software architect is constrained from the start of the design phase to choose a platform in which to implement the system.

1.2 Scope

This thesis is all about models, the language they are based on, the editor they are constructed in and the concepts that are included in the model. It has a wide scope, spreading from different conceptual modeling architectures like MDA or SOA, design methodologies as in LDD or MDSD, and narrowing down to metamodels for a specific domain. Still concerning the model, it goes deep into GMF, which is used to implement a graphical editor for a modeling language .

1.3 Objectives

The main objective of this thesis is to provide a prototype in which we could design SOA-based systems using a DSL. This prototype is a graphical modeling editor, based on a given metamodel. The name of the thesis reflects the main objective as we named it for Metamodel-based Editor for Service Oriented Architecture (MED4SOA).

Through the development of a solution that should achieve the main objective, the thesis will investigate the following hypothesis:

H1 *A customised graphical editor for a high level DSL will be an efficient tool to design SOA-based systems.*

Based on this hypothesis we will test the efficiency of MED4SOA, meaning the ability of this graphical editor to provide correct and understandable models.

Under the development of MED4SOA and the investigation of the hypothesis, we aim to identify the two main parts of the editor, the metamodel and the graphical framework. In addition to the identification, we aim to investigate further the metamodel, revealing its capabilities or the need of change. Similarly, the capability and efficiency of the graphical framework will be explored.

In the entireness of this thesis, we aim to provide an introduction to the state of the art of metamodel capturing the service domain, and generative graphical framework, which realize DSLs based on these metamodels. Our goal is to provide a new approach in the construction of graphical editors and to lay the fundamentals of designing an SOA using a DSL for services.

1.4 Research method

In this thesis we apply a research method known as *technology research* [6]. This kind of research method concerns the development of a new *artefact* or improvement of an existing one. The new developed artefact, is MED4SOA.

Following the technology research, this thesis conducts a problem analysis that points out the need for a new artifact. Earlier in the problem analysis we introduce an *initial case study*, based on an e-procurement scenario. This case study will help us to identify the problem(s) and later serve as modeling example to illustrate the features of MED4SOA.

In technology research an overall hypothesis is formulated in order to show that the *artifact satisfies the need*. Based on this hypothesis we specify the requirements for its two main parts and conduct a theoretical research in the state of the art for a metamodel capturing the service domain and a graphical editing framework.

The overall hypothesis is tested by predictions. Given the aim of this thesis, our hypothesis is tested via 7 predictions, which we called for success criteria. The evaluation of the success criteria will later in the thesis validate the hypothesis.

The evaluation of the success criteria and the evaluation of the technologies used in the implementation of the graphical editor, is undergone through test strategies, defined in technology research as *non-empirical evidence* and *laboratory experiments*. The non-empirical evidence implies analytical argumentation, high generalisation and low precision, while the laboratory experiment gives developers the opportunity to test their systems. These strategies combined with the initial case study, are used in this thesis to reach the appointed objectives.

Going from the research method to software development techniques, we should mention that in the development of MED4SOA we use a software engineering technique, known as software prototyping [7]. With a prototype, we should have a rapid development of MED4SOA and experiment with this solution in order to demonstrate its concepts, or find out more about the problem and its possible solutions.

1.5 Structure of thesis

This master thesis has the following structure:

Chapter 1 introduces the motivations and objectives of the thesis. It also gives a description of the research method applied to achieve the objectives.

Chapter 2 contains the background for the research carried out in the thesis. It introduces concepts, paradigms, initiatives and standards used in the development of the prototype.

Chapter 3 conducts a problem analysis to identify the need for the artefact that will be developed. A case study is introduced early in this chapter. Further, a solution is proposed and the hypothesis is stated. This chapter also lists the requirements for the main parts of the solution, and the success criteria that the hypothesis is validated upon.

Chapter 4 investigates the state of the art technologies in the field of metamodels for the service domain and graphical modeling frameworks that realize visually these metamodels. It shortly introduces these technologies, and later in the chapter, evaluates them upon requirements listed in the problem analysis. The evaluation results will indicate the most suitable metamodel for services and the graphical framework that will implement this metamodel.

Chapter 5 describes in details the architecture of MED4SOA. It will start with the description of an abstract architectural overview, continue in the description of the structure and concepts of the metamodel and further on the description of the technologies involved in the implementation of the editor. Later, the chapter is introduced the development process for constructing the editor.

Chapter 6 concentrates on the implementation and realisation of MED4SOA. It will cover several aspects of the implementation, starting with the implementation of the metamodel, continuing with the creation of the graphical elements, tooling, mapping definitions and concluding with the generation of the editor. Aspects of implementation of constraints in the metamodel are also described in this chapter.

Chapter 7 provides an example of use of MED4SOA, based in the case study that was introduced in chapter 2. Through the example, this chapter shows the ability of the editor to provide graphical modeling facilities that can create models concerning the information, service or process aspect of a SOA-based system.

Chapter 8 starts by evaluating the fulfillment of the success criteria and provides a short argumentation on the result of the evaluation. It engages further into a discussion about the metamodel and the graphical framework. The discussion points out our experience in the development of MED4SOA, recommendations in the future development of the metamodel and different characteristics of the graphical framework.

Chapter 9 summarizes the work done in the development of MED4SOA. It lists the achievements of this master thesis and provides recommendation on future work.

Appendix A provides an Ecore model of PIM4SOA .

Chapter 2

Background

"Not to know what has been transacted in former times is to be always a child. If no use is made of the labors of past ages, the world must remain always in the infancy of knowledge."

Marcus Tullius Cicero

MED4SOA relies on different aspects of software development. It shares concepts connected to different architectures or approaches in the development of a new artefact. In this chapter we will introduce these concepts and approaches aiming to provide a basic knowledge in the background of MED4SOA.

The sections to follow, will introduce MDA, including description of concepts related to metalevels, levels of abstraction of the model and the importance of the model itself. It will introduce MDSD and the view this approach has on DSLs. LDD is introduced later in the chapter, while the last section will focus on SOA, aiming to describe it basic concepts, characteristics of the service and how this concepts are connected to information technology.

2.1 Model Driven Architecture (MDA)

MDA is an OMG initiative, which defines a model driven approach to system development [8]. Its main goals are interoperability and portability, respectively aiming to achieve an independence from the manufacturers and independence from the software execution platform. In order to achieve these goals, MDA provides guidelines and standards that will lead to the structuring of specifications in form of models.

The model is the centerpiece of MDA's mentality. It is the model that describes visually the abstract, physical or hypothetical reality. It is the mean of communication and the starting point for a automated transformation. A model is based on a modeling language.

In the MDA prospective, the language of choice is UML[1]. This modeling language is the de facto standard and the most used language in the design of software systems [9]. UML is a commonly used modeling language in the context of MDA. Anyhow MDA does not have any terms against the usage of DSLs. In longer terms, DSLs are more likely to be an important alternative to UML.

Both UML and DSLs are defined in a metamodel which in its turn is an instance of a metametamodel. MDA defines a standardized terminology involving a four-level metamodel hierarchy.

2.1.1 Metalevels

The four metamodel levels are defined by MDA and OMG to ease the comprehension of the abstractions involved in modeling. Figure 2.1 describes these levels using a *non-formal* concept of a service.

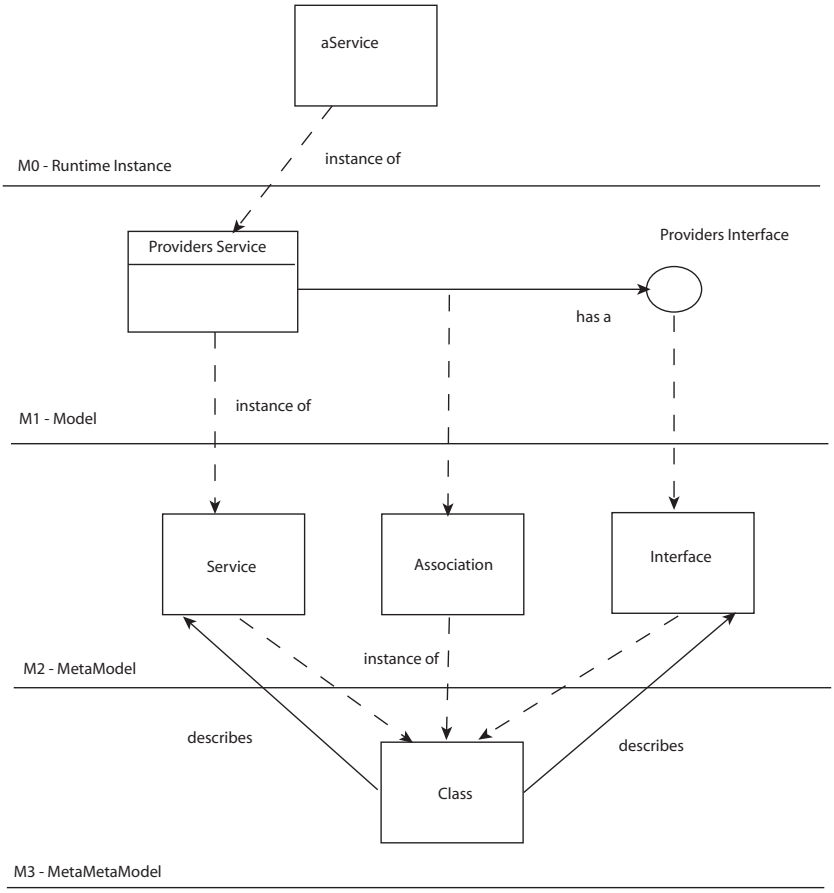

Figure 2.1: The four-level metamodel hierarchy

[1]The present(2007) version is UML 2.0

The four levels described in the figure above are:

M0 contains the instance of a model. It is the data used in an application, usually at a program runtime, e.g. the implementation of a service in a object oriented system.

M1 contains the model. We are familiar with models that are constructed in the design cycle of the development process. These models will describe the system that will be implemented. In the figure above we show the model of a service which is associated to an interface.

M2 contains the metamodel. A metamodel is the model of a modeling language, where all the concepts and mechanisms of this language are described. It describes the structure, semantics and the constraints of a model. The elements of M1 are defined from the metamodel, e.g. service, interface, association.

M3 contains the metameta-data. This is the highest level of abstraction and it is self-defined. The metadata contained in this level captures the modeling language. MDA provides a standard metamodeling facility, Meta Object Facility (MOF), which provides the abstractions to define UML or other DSLs.

Another important aspect of MDA is the transformation between model. According to the MDA paradigm the development process is carried out by transforming conceptual models in different levels of abstraction. MDA defines three main levels of abstraction, which we describe below.

2.1.2 Levels of abstraction

In order to achieve the independence from the software application platform, and for reasons of longevity in the rapid change of the business development, MDA defines three main levels of abstraction. These are:

- The Computional Independent Model (CIM) focuses on the environment of the system and hides the structural details regarding the implementation platform. It captures the business context and the business requirements. Usually, the CIM is constructed by a business analyst.

- The Platform Independent Model (PIM) describes the system from a platform independent perspective. It captures the abstractions of one or more platform, by hiding the platforms specific data. The platform in this context is the set of technologies and subsystems that provides the needed functionality. A PIM is mostly used in describing the architecture of a system, including its operation and details, but without any specific implementation data.

- The Platform Specific Model (PSM) is the representation of the system including the platform specific data. It combines the specification of the PIM with the details of a specific platform. In the MDA context, a PSM is created via a model transformation from a PIM. For instance, a PIM describing the domain model

of a system can be transformed to a specific implementation platform like J2EE [10] or .Net [11].

These are the three main abstraction levels described by MDA. To note here is the relativity of PIM and PSM, depending on the defined platform. A PSM may be a PIM in respect to some implementation details, e.g. a model may be specific for J2EE, but still a PIM according to the database implementation details.

2.1.3 Diagrams vs. models

A common misunderstanding in the modeling world is the equality between a diagram and a model [12]. A diagram is not a model. It is a representation of some aspect of the model, using visual representation like lines, nodes or shapes. According to the OMG way of thinking, the model is the whole machine-readable description of the system. Following this way of thinking, a modeling tool should not be only about the manipulation of a diagram, but it should also conform to the model and its metamodel.

To ensure the manipulation of the model, and even down to the metamodel level, MDA defines a set of standards for support of data access. One of these standards is the XML Metadata Interchange (XMI), which is the backbone of the MDA infrastructure. XMI defines rules for deriving eXtensible Markup Language (XML) schema from a MOF-based modeling language. It is the basis for interoperability between MDA modeling tools. Even though created to serialize UML models this standard is widely used also for DSLs.

Concluding this section on MDA we should mention that in software development the MDA approach is known as the Model Driven Development (MDD) [13]. In the MDD the model is not a sketch or a blue print. They are the primary artefacts from which efficient implementations are generated by the application of transformation. We will focus in the next section in a flavour of MDD, called MDSD.

2.2 Model Driven Software Development (MDSD)

MDSD is a MDA based approach where the **model** does not constitute the documentation of the system, but is regarded as equal to code [2]. It is believed [2] that MDSD is quite important for the future of software development. It could be the natural continuation of programming as we know it today.

The MDSD approach differs from MDA mainly due to different motivations. Being a OMG initiative, MDA promotes the usage of UML in software development, while MDSD have a different view on it. MDSD promotes the application of customized modeling languages, DSLs, assuming that not everyone has the same needs on a given

[2] by the authors of [2] and us the followers/believers

domain. Therefore it aims to find domain specific abstractions that capture the concepts of a domain and make them accessible through modeling.

MDSD focuses in the practical application of the models in software development. It aims to provide modules in software development processes that are used in the context of model-driven approach independently of the standard or selected tool. This practical application has still the model as *a part of the software*, but in contrast to MDA the model should be seen as the final bulk of implementation. The model should not only describe the method skeletons, but it should be the starting point of automated transformation to the actual implementation. To do this a formal model is needed.

In order to ensure this formalization of models DSLs are required. We will explain further the need of a DSL and the tools that support it.

2.2.1 Domain Specific Language (DSL)

A DSL captures the abstractions of a domain. DSL could be of different type, textual or graphical. In the context of this thesis we will focus on the graphical modeling language and as such it is based in a metamodel. The DSL has its abstract syntax and semantics included in the metamodel, while the concrete syntax which represents visually its concepts is defined separately. More on the syntaxes and the semantic of a modeling language is provided in section 2.3.

DSLs are applied in the construction of graphical models. A primary benefit of a DSL is its ability to provide abstraction tailored for a specific type of a domain [14]. An important aspect of a DSL is the *domain*, by which we understand a collection of related concepts that is specific to a particular domain. Therefore a DSL is the modeling language which provides mean of expressing these concepts graphically.

In order to be useful in the software development process a DSL needs tool support. The tool in MDSD context is a suitable editor for the specific modeling language.

2.2.2 MDSD tools

In MDSD it is essential that good editors are provided. There are many tools playing different role in this approach, but in the context of our theses, we focus on the DSL editors. Depending on the type of the DSL, there are two classes [3] of DSL editors, UML editors(a DSL through a profile) and custom-made specific tools.

UML tools

UML tools are used to create UML models. Except the standard diagrams, UML gives the opportunity to extend its vocabulary through *stereotypes*. Stereotypes are

[3] there are also DSL editors for textual DSLs but they are beyond the scope of the thesis

defined by UML profiles, which are language concepts defined via basic UML constructs. They extend the UML language providing concepts for a specific problem domain.

Many of these tools are listed by OMG [1] and most of them are able to deal with UML profiles. They use XMI standard to store the model, while stereotypes are added to the usual UML elements to tag the domain specific elements. Code generation and correctness check with the metamodel are some other important features of these editors. A representative of the commercial industry for these tools is the IBM's Rational Software Modeler (RSM) [15] while an open source candidate is the Eclipse MDT project [16] and its UML 2.0 tools.

Customized DSL tools

Customized DSL tools create models conforming to a specialized DSL. The metamodel of a DSL is implemented in such a tool. DSL tools can construct or read input models and check them for correctness against the metamodel. As we are focused on graphical DSLs, graphical editors are the tools of choice, but there is still considerable effort to create them [2].

To achieve a greater paste in the software development, even not mandatory, a generated DSL tool is preferred. These specific tools are created by generative frameworks that provide the facilities to implement the metamodel, the concrete syntax of the language and generate the editors to support it. Even though this kind of technology is not mature yet, in the near future these editors could play an important role in MDSD.

2.3 Language Driven Development (LDD)

LDD is another approach to software development which is focused on the language and its tools to capture the software solutions [17]. This approach aims to raise the productivity by concentrating on powerful language abstractions and development environments that support the languages and processes. It is fundamentally based in the ability to rapidly design languages and their supporting tools, based on metamodeling.

LDD is a model-driven development approach based in MDA. Its ideas complete the MDA philosophy, by enabling all aspects of the development process to be captured as models. According to LDD it is the language which provides the solution and together with a powerful LDD framework it will allow the construction of agile abstractions that are resistible to change. In this context, the distinguishing from MDA relies in the inclusion of the language artifacts in the development process. A model should not only have the focus in the high-level abstractions it contains, but also *in the modeling language, which is in itself the main abstraction* [17].

In the construction of a DSL there are three main features: the abstract syntax, concrete syntax and the semantics of the future language [17] [2]. From now on when we describe a language, we will refer to a graphical modeling language.

2.3.1 Abstract syntax

The abstract syntax of a language defines the vocabulary of abstractions used in this language. The abstractions are the definitions of the elements that combined together will form the model. The abstract syntax is described in the metamodel. In the metamodel we define the structures of the language, their relationships and the rules that define the well-formedness of the model toward its metamodel.

The structures of a language are any definition of the concept used in it. For instance, classes, attributes, references are the structures of UML. The connections between these structures form the relationships of the abstract syntax and rules are defined to constrain the use of the language. These rules are useful in the case of a tool support for the language, since they can be used in the validation of the models. Mostly, the well-formedness rules are written in Object Constraint Language (OCL).

2.3.2 Concrete syntax

The concrete syntax is the set of notations that facilitates the visualisation of the language constructs. It is the representation of the abstract syntax of the language. The concrete syntax is also referred to as the visual syntax.

The concrete syntax represents the model in a graphical way. For instance, UML uses *nodes* and *edges* to represent some underlying model elements, while other DSLs may have different notations. The notations are visualised through figures, e.g. nodes in UML by rectangles and ellipses. The main benefit of a concrete syntax, is its ability to represent the abstract syntax in different ways.

2.3.3 Semantics

The semantics of the language define the meaning of its elements. A semantics definition is a very important feature in the language. It clarifies what the actual elements represent and leaves no place for assumptions.

The semantics is a key element in order to understand how to use a language. In the context of a DSL, even if the developers may have an understanding of the concepts of the problem domain, the semantics are the key to clarify these meanings. They also have a central role in the definition of *semantically reach capabilities, such as execution and transformation that are required to support MDA and LDD* [17].

2.4 Service Oriented Architecture (SOA)

The last book on the subject that came to our attention, introduced SOA as *a concept whose time has come* [4]. SOA is not new, it has been around for many years, but it has gained popularity due to the increase of the complexity in today's software systems. It is not a "fix" to this complexity, but a vision that allows the IT-functionality to be

delivered as modular business services in order to achieve specific business benefits. But what is actually SOA.

There is no standardized definition of SOA, but according to [4], it is a conceptual business architecture that allows business functionality or application logic to be available through reusable IT services. It is a concept, a philosophy, an approach of how IT functionality can be planned designed and delivered in such a way that it will achieve a specific business goal. Therefore it assures interoperability, reusability and integration across all business processes and technology platforms.

According to the OASIS [4] SOA Reference Model [18], SOA is a paradigm for organizing and utilizing capabilities of different ownership domains. It provides the means of organizing solutions owned by others, which combined with locally "owned", enable a more valuable usage of these solutions. The main concept of SOA is the **service**. It is the centerpiece of SOA and considered also as its primary architectural asset. We will base our description of the service and other relating concepts in the reference model introduced below.

2.4.1 SOA concepts

The Reference Model for SOA is an abstract framework for understanding significant relationships among the concepts of the service domain. In the absence of a standard, the abstract framework aims to provide the essence for SOA, in a vocabulary and a common understanding of its concepts. It provides an abstract model not connected to any various existing or future deployment technology for this architecture. Figure 2.2 shows the concepts of the SOA Reference Model. To reduce the complexity of the figure, we did not show concepts related to the **execution context** or **real world effects**.

A service is described in the reference model as a mechanism that enables access to a set of capabilities. The capabilities incorporate the business behaviour, while the access to them is provided using prescribed **interfaces**. The service is provided by a **service provider** and invoked by the *service consumer*. Except the **service description** that specifies the capabilities offered and the information needed to use the service, the rest of the information is hidden to the service consumer.

In the context of service dynamics, there are some other concepts involved in the interaction between services: the **visibility** between exchanging parties and the actual interaction between them. The visibility relies in the ability of providers and consumers to interact with each other, while the interaction itself involves performing actions against the service. In many cases the interaction takes place through message exchange.

In order to describe the interaction process, the service description references an **information model** and a **behaviour model**. The information model of a service describes the information that will be exchange with the service. It includes both the structures(syntax) and meaning(semantics) of the information to be exchange. The

[4]Organization for the Advancement of Structured Information Standards

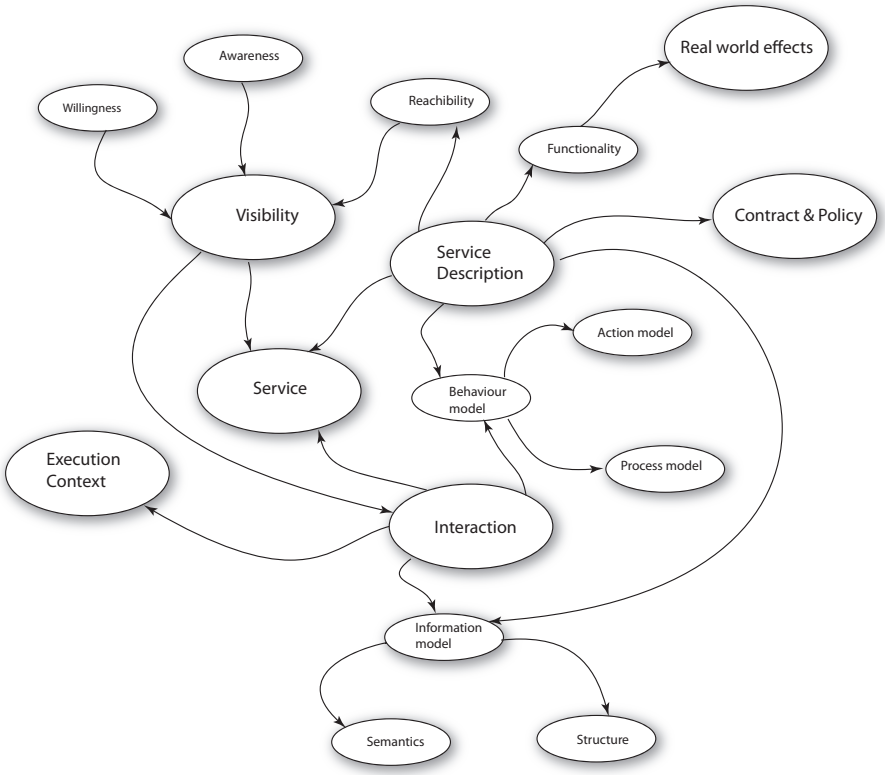

Figure 2.2: The concepts of the OASIS SOA Reference Model

behaviour model provides the knowledge of the actions invoked against a service, including the process of interacting with the service. The interaction itself is described in the **execution context**. In the execution context are defined the set of infrastructure elements, processes and entities that form the path between the service provider and consumer.

Other concepts about the service are the **service functionality**, included in the service description and **policies & contracts**. The policy represents some constraints or conditions on the usage of the service, while the **service contract** is the agreement by two or more parties.

After introducing the service and other concepts related to its description or interaction, we will have a look at some of its characteristics.

2.4.2 Characteristics of a service

The service, being the primary asset of an SOA, has some characteristics which provided the needed value to the organization that will adopt it [4]. These characteristics are also seen as attributes of SOA and take an important place in this paradigm. Below we describe some of the characteristics of the service.

Coarse grained services is one of the most used words in SOA. This characteristic involves the functionality the service encapsulate. The service represents business functions and processes. Other components or services are also contained within functions and processes. The granularity depends on the functionality of the service. For instance too coarse-grained services will be "heavy" and encapsulate a large number of components and transactions, while fine-grained services have a narrow scope and fewer elements.

Well defined service contracts involves having a clear specification on the functional and technical details needed to consume a service. It gives the outside world the needed information to use the service, while hiding the technical details.

Loosely coupled services is a term that in the context of service design, implies for the service to have a specific implementation. This characteristic describes the service as an entity not bound to the implementation technology. In the need of change this implementation could be replaced without any further changes in the architecture of the system.

Discoverable in the service aspect means that the service constructs or service description except for being well designed should also be published and visible to the consumer. They should be easily discovered implying that the services are listed in service registries in order to achieve a bigger audience than the one intended in the design.

Composable is a term use in SOA to characterize a service as a collection of other components or services. The service contains the components that form its functionality, but other services may also be incorporated in it. This leads to an incremental extensibility of services.

A service in SOA is considered to be **business aligned**. It should encapsulate the business requirement, behaviour and other imperatives which are found in business modeling. In order have a long lasting alignment, the service should be **durable**. The durability of the service do not imply it to be rigid and not flexible, but to hold the relationship to the business it represents.

The service in SOA is **reusable** and **interoperable**. These are the main values that SOA promotes, but also the most difficult to achieve. The reusability implies incorporating the proper functions of the services, in the way that it can have a clear definition of its use and multiple consumption partners. The interoperability involves clear definition of the application policies and standards of the services.

2.4.3 SOA modeling and implementation

In the context of the implementation of an SOA, three are the major levels of abstraction: *operations*, *services* and *business processes* [19]. Starting from the low level of abstraction, the operations represent the transactions as a single unit of work. They could be comparable to Object Oriented(OO) methods. The services in a SOA model represent the logical grouping of these operations. The business processes have the

highest level of abstraction, hence including actions and activities with specific business goals. Business processes may include multiple services.

The description of SOA we provided above, was more focused in the abstract definition of its concept, features and characteristics, but the popularity of SOA relies in its practical implementation in software systems. The levels of abstraction we described represent and lead to an implementation technology. The technology used mainly in implementing SOA is Web Services [20] [21].

Web Services [5] are the set of protocols by which services can be published, discovered and used in a standard technology neutral form. This definition may seem inaccurate in respect to the description of abstraction levels introduced earlier in this chapter. The inaccuracy relies in the presentation of web services as a technology neutral, implying a PIM. But as mention earlier, the platform independency is relative. From our point of view, we will consider Web Services to be a PSM. Other technologies like Agents, Grid, Peer to Peer (P2P) and Semantic Web Services are also alternatives to SOA implementations.

2.5 Summary

Here we conclude the background chapter, which introduced three important aspects of the concepts behind MED4SOA. In the introduction of the MDA approach, we pointed out the distinction of abstractions in the four-level metamodel hierarchy. We also described the different levels of model abstraction, CIM, PIM, PSM, and pointed out the difference between diagrams and models.

MDSD was introduced as a flavour of MDA, focusing on the practical application of models in software development. We described the relation of MDSD to DSLs and the need for tool support that this approach intends to provide to modeling languages. Continuing with software development approaches, we introduced LDD which sees the language and the tool that support this language to be the solution. LDD explained the three main features of a DSL, being the abstract syntax, concrete syntax and semantics.

The last section of this chapter was focused on SOA. In this section we described its main concepts, including the service as the centerpiece and concepts of service interaction or service description. Later in this section we introduced the main characteristics of a service and concluded with the relation of SOA to models and implementation technologies.

Chapter 3

Problem analysis

"Man is not born to solve the problems of the universe, but to find out what he has to do; and to restrain himself within the limits of his comprehension."

Johann Wolfgang von Goethe

THE problem analysis conducted in this chapter aims to provide a clear picture of problems that Small and Medium Enterprises (SMEs) face in the rapid change of business innovation. We will focus on the IT aspect of the problem and investigate the interoperability issues in this context. A case study, provided in the next section aims to show the alignment and interoperability between business and technology. The main concepts of this case, comprising services, processes, activities and information will be later modeled in chapter 7.

The problem analysis will characterize issue(s) from the case study and propose a solution. A hypothesis will be stated, and two sets of requirements will be listed. Alternative technologies investigated in chapter 4, will be then evaluated on the basis of these requirements.

The problem analysis will be concluded with the definition of the success criteria. The success criteria are use as predictions to evaluate the hypothesis in chapter 8.

3.1 Case Study: e-Procurement

In this section we provide a case study, which will help us identify IT-related problems in the furniture industry. It investigates the situation of SMEs that are willing to profit from e-business automated systems. The case gives a clear picture of the enterprise and the activities within it.

The AIDIMA [1] e-procurement scenario is based on the current situation of the furniture *SME*s (i.e small companies). It was proposed on I-ESA 2006 [23] as an innovative

[1] AIDIMA is the RTD Association of the Furniture, wood, packaging and Related industries technology institute in Spain [22]

scenario that facilitates the interoperability among e-Business services, and the respective implementation of the integration mechanisms in SMEs-based business scenarios [24].

The AIDIMA scenario focuses in e-Procurement activities. Three actors are identified in the process: Retailer, Manufacturer and Provider. A Retailer may be a small selling company, which sells furnitures to the public population. A Retailer furnishes from a Manufacturer, which in its turn get supplies from a Provider. The scenario is divided into two parts: Customer-oriented and Supplier-oriented sub-scenario. The next sections will give a detailed description of the sub-scenarios.

3.1.1 Customer-oriented sub-scenario

The customer-oriented scenario reflects the intention of the customers to purchase goods from multiple suppliers. Fig 3.1 presents the document flow between a Retailer and a Manufacturer. The document flow starts when a Retailer requests some furniture from a Manufacturer. A buyer(Retailer) sends a Request for Quotation (RFQ) to a supplier(Manufacturer). As soon as the RFQ is processed, the supplier sends back his Quotation to the buyer. The buyer on his side evaluates the Quotation and, eventually, processes the Order. When the order arrives at the supplier it will be accepted, fulfilled and the goods will be delivered to the consumer. The supplier finalizes his part by sending the Invoice to the buyer. The buyer than concludes the sub-scenario by paying the Invoice.

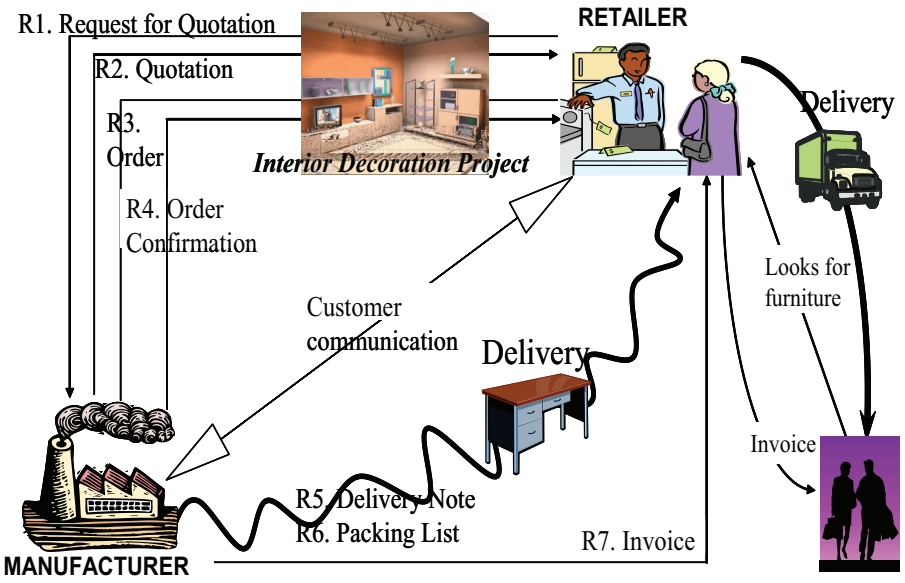

Figure 3.1: Selling process: Customer-oriented scenario

The documents utilized in this scenario are labeled as : R1. RFQ, R2. Quotation, R3. Order, R4. Order Confirmation, R5. Delivery Note. R6. Packing List, R7. Invoice.

3.1.2 Supplier-oriented sub-scenario

The supplier-oriented sub-scenario puts the Manufacturer in the buyer's position. In this case the Manufacturer will acquire parts from a Provider. When an order is issued from the consumers' part and an order confirmation is sent as a response, the Manufacturer sends a RFQ to one or more Providers. The Provider responds back with a Quotation and awaits an Order. An Order is issued from the Manufacturer after the Quotation is considered and found optimal. The Provider that receives the Order responds with an Order Confirmation. When the time comes, the Provider will deliver the parts together with a Delivery note and Invoice. Fig 2.3 shows the scenario and the information flow between the Manufacturer and the Provider.

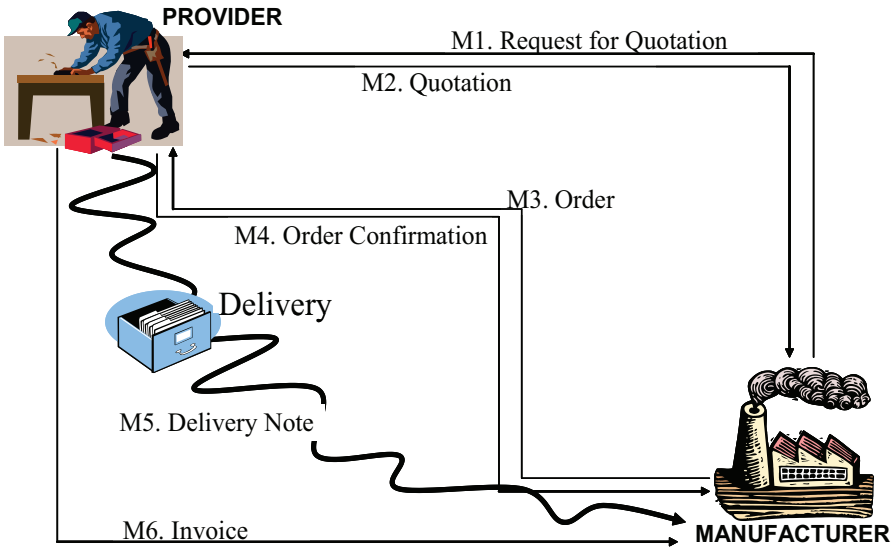

Figure 3.2: Procurement process: Supplier-oriented scenario

The documents used in this part of the scenario are: M1. RFQ, M2. Quotation, M3. Order, M4. Order Confirmation M5. Delivery Note M6. Invoice

3.2 Problem description

In the case study described in the previous section, we could recognize several interoperability related issues. They include repetitive processes for orders, confusion resulting poor product description, missing information both from supplier and provider or extended time usage in delivery [24].

The furniture industry in its complex structure, needs also to be able to dynamically adapt themselves in order to take advantage of marked opportunities [24]. These marked opportunities, included business profits, could be fully achieved through the alignment and interoperability between business and technology. Due to their small

size and limited resources, SMEs do not have the facilities to develop a solution that could adapt their business procedures and be interoperable with others. The challenge in this case should be in how to provide a solution that solves the e-business integration.

Given that e-business is based on services provided by some parties and consumed by others, the problem for providing a solution to the AIDIMA industry relies in the non-existence of a modeling language and supporting tool for this domain. Indeed there exist UML modeling frameworks like RSM [15], which provide features to model general systems, but there is no editing framework for a SOA DSL. Using SOA principles, we could provide a model that will describe the e-procurement scenario. This model could be constructed in a Metamodel-based Editor for Service Oriented Architecture (MED4SOA). The next sub-section introduces the proposed solution.

3.2.1 Proposed solution

In the previous chapter we looked at LDD, suggesting that developers could use powerful language abstractions and development environments that support their developed processes and languages. The new approach implies that metamodeling is the way to achieve this. Reckoning the metamodel to be the model of a language that captures its essential features and properties, we will concentrate in finding a suitable metamodel of a modeling language for the service domain.

A formal metamodel capturing the service domain is a good starting point, but to have a productive value this metamodel needs a custom, domain specific Integrated Development Environment (IDE). This environment in the context of this thesis would be a graphical editor which main purpose is to deliver a service model that conforms to the basics and concepts of the defined metamodel.

A traditional approach to build a SOA-based system would be applying a design pattern and implementing the solution in the state of the art standard, Web Services, or other technologies. The alternative approach we propose is to combine the principles of SOA, LDD, MDA and MDSD, focusing in the design of the system. Hereby, we propose the solution in MED4SOA, being a graphical modeling editor in which architects can design a SOA-based system independently of the implementation platform. The final implementation of the system could be reached by applying MDA principles, namely transformation of the models provided by MED4SOA.

Our assumption about the solution, needs an testable hypothesis. The hypothesis is stated in the next subsection.

3.2.2 Hypothesis

While in the previous part of the problem analysis we described a case study, recognized the problems within its domain and proposed a solution, in here we state the hypothesis that this theses needs to address:

H1 *A customised graphical editor for a high level DSL will be an efficient tool to design SOA-based systems.*

In order to investigate further the hypothesis we have to find a metamodel created with the purpose of abstracting the concepts of the service domain. Further, this metamodel will be used as a basis for creating the graphical editor. In this context, it is important to find out:

1. What is the most suitable metamodel that can be used to define this modeling language?

2. What editor framework could best support the specific language?

We will find the answers to these questions by providing a set of requirements for a SOA metamodel and editor framework. The alternative technologies described in chapter 4 will be evaluated upon the basis of these requirements and the most suitable metamodel and editor framework will be chosen in implementing MED4SOA. The requirements are listed in the following sections.

3.3 Requirements for a SOA metamodel

In the fall of 2006, OMG launched a Request for Proposal (RFP) on a UML Profile and Metamodel for Service (UPMS) [25]. Based on MDA paradigms, one of the scopes of this RFP is to facilitate the adoption of SOA through more abstract and platform independent models. In order to achieve it, the RFP requests a common vocabulary and metamodel to unify the diverse service definitions that exist in the industry.

The UPMS RFP introduced above, request a metamodel and profile for extending UML2.0 with capabilities relevant for modeling services using a SOA. In this section we will disregard the UML profile and concentrate on the metamodel. The RFP provides a set of requirements for the service metamodel, in which we partly base our list of requirements provided below. The requirements are also based on the definitions provided by the Organization for the Advancement of Structured Information Standards (OASIS) SOA Reference Model for Service Architecture [18]. Table 3.1 gives an overview of the requirements. We named the requirements MR1-7, with MR being the abbreviation of metamodel requirement.

The metamodel shall:

1. be based on the facilities provided by MOF.

2. provide the platform independence needed in modeling enterprises. It shall express the intent of service models rather than any specific means by which that intent may be realized by some specific

	Requirement
MR1	MOF-based
MR2	Platform Independent
MR3	Business-oriented
MR4	Loose coupling
MR5	Expressiveness
MR6	Extensibility
MR7	Simplicity

Table 3.1: Requirements for a SOA metamodel.

3. provide concepts describing business-oriented modeling elements, which permit the extraction of sufficient information to describe the enterprise. It shall provide the fundamental building blocks that combines the design of information and behaviour, comprehended by both business and modeling specialists.

4. have the ability to give the necessary abstraction of the service concept. The service shall be a self-contained software block incorporating the business behaviour and not connected or limited to a standard or technology.

5. provide means of designing the main concepts of the service domain.

6. have the ability to allow additional specifications of concerns like security, QoS, manageability, etc.

7. be as simple as possible, but still providing the means to model the three major levels of abstraction within SOA, *operations, services and business processes*.

3.4 Requirements for a graphical editor

Stahl and Voelter implied in their book [2] that MDSD does not make sense without tool support. For working with a visual modeling language for services, a graphical editor is the tool of choice. Either way, if we choose a UML profile or other notations to visualize the elements models, there are a set of requirements that these tools shall meet, in order to satisfy the industry needs. Inspired in the MDSD approach we provide a list of requirements for a graphical editor, which will be based on the service metamodel. Table 3.2 gives an overview of the requirements. The requirements in this part are named GFR1-7, being an abbreviation of graphical framework requirement.

The graphical editor shall:

1. be implemented in an open source technology. [2]

[2]This requirement is chosen in order to support the SMEs and not inflict any other cost on the development process

	Requirements
GFR1	Open Source technology
GFR2	Metamodel-based
GFR3	Flexibility
GFR4	Low implementation effort
GFR5	Isolation of notations
GFR6	Extensibility
GFR7	Multiple view support

Table 3.2: Requirements for a graphical editor.

2. be metamodel-based. The tool shall not only realize the metamodel elements graphically, but also check for correctness against the metamodel.

3. provide the sufficient flexibility to allow the description and representation of all the required information.

4. have a low implementation effort in order to ensure a high paste in the development process. Automatic generation of editors is not required, but should be considered.

5. provide the isolation of the graphical or editor notations from the underlying semantics provided in a metamodel.

6. be an open editor which architecture and constructs could easily be extended by the developer.

7. have a graphical interface that could design and manipulate visual models in different views.

3.5 Definition of success criteria

Since the hypothesis, **H1**, is stated, and the requirements for its main parts listed, the next step would be to define the success criteria by which we will test the hypothesis. The success criteria will focus on the expectation of **H1**, meaning the *efficiency* of designing SOA based systems. The word *efficient* may have different definitions involving rapidity, energy spent or wasted [26]. In our context it does not define any measurable means. *Efficient* is an adjective describing *a productive of the desired effects* as defined in [27].

Through the success criteria we define the desired effect that MED4SOA will provide. They are presented as predictions, and will be evaluated in chapter 8. This evaluation will validate the hypothesis.

3.5.1 Clarification of specifications

Before we list the requirements we want to specify some concepts that will clarify the specifications used in the success criteria and later in the thesis.

User

The user in MED4SOA is a business analyst that will model the solution that he/she is willing to implement. A user can be also a system developer, whom is cooperating with the business analyst in the enterprise. We will address both business analyst and system developer as a *user* of MED4SOA.

Data

There are two aspects of data stored in MED4SOA, notations and semantic data. Notations represent the diagrammatical data that include shapes, nodes, connections, compartments etc. The notations are the data that will be displayed in the editor. From the users point of view there is no difference between notations and the semantic data. Notations are a simply a window to the semantic data.

The semantic data on the other hand, represent what the user will be editing. These are the elements that are stored in the metamodel. Usually, the semantic data of a model is bound to a pre-specified graphical notation, which in the case of MED4SOA could be different type of nodes or connections.

Elements

In the success criteria and in the next chapters to come we will refer to three different kinds of elements: domain, graphical and diagram element. The domain elements are the abstractions, concepts or metamodeling mechanisms include in the metamodel. The graphical elements are the figures that will represent the domain elements, while the diagram elements are the graphical elements that are present in a diagram. They may be nodes, links, labels and so on.

3.5.2 Success criteria

The Metamodel-based Editor for Service Oriented Architecture (MED4SOA) shall:

Success criterion 1

P1: *provide means to construct models **graphically**.*

Success criterion 2

P2: *realize graphically **all** the concepts and abstractions forming the semantic data in a SOA metamodel.*

Success criterion 3

P3: *provide means that will make possible the representation of the semantic data in different contexts. These contexts could be different views or graphical representation of the concrete syntax.*

Success criterion 4

P4: *be applied to create persistent models that have the ability to display semantic data multiple times in the same diagram.*

Success criterion 5

P5: *promote simplicity in the design process. The simplicity covers both the aspect of comprehension and structure of the models created.*

Success criterion 6

P6: *represent the domain elements in conformance to their semantics and notations applied in the modeling world.* [3]

Success criterion 7

P7: *provide the user with clear and distinguished diagram elements in the model.*

3.6 Summary

In this chapter we conducted a problem analysis in which the rest of the thesis will be based on. We started with describing a case study, from the AIDIMA e-procurement scenario, from which we identified problems and needs industry. Then we gave an overall problem description and stated the hypothesis for this theses. Further, in order to investigate the hypothesis, we listed two sets of requirements, concerning the metamodel for SOA and the editor framework. In the end, we clarified the specification that will be used further in the thesis, and defined the success criteria, being the predictions that will test the hypothesis.

[3] in conformance meaning that for instance a association will be a kind of a link and not an airplane flying from one node to the other

Chapter 4

Technologies

"Once a new technology rolls over you, if your're not part of the steamroller, you're part of the road."

Stewart Brand

In this chapter we introduce different technologies in which MED4SOA will be based, and evaluate them in order to find the most suitable. The chapter is divided in three parts. The first part introduces three metamodels that describe concepts of the service domain. We will give a short description of their basic concepts and the standards they are build upon.

The second part introduces candidate frameworks in which to implement MED4SOA. These frameworks support construction of editors in which the metamodel concepts will be visualized graphically. For each editor framework we will present their architectures and main features.

The third part provides an evaluation of the different candidates, based on the requirements listed in the previous chapter. The evaluation is based on scores which we define later in the chapter.

4.1 Metamodels for SOA

We mentioned earlier in chapter 2, that the concepts of a language are defined in its metamodel. In chapter 3 we characterized our problem and the solution proposed indicated the construction of a graphical editor in which we could model, based on SOA concepts. These concepts are specified in a metamodel for SOA.

In the next sections we will look at three different approaches that describe the concepts of SOA: the IBM Service model, UML for EDOC and PIM4SOA. We chose these approaches not only because they represent the state of the art in service modeling, but also due to the providers behind them, going from one of the biggest commercial providers inside modeling technology to the largest computing industry standards group.

4.1.1 IBM Service Model

The IBM service model [28] is a conceptual model that allows designers and architects to specify services and their capabilities in a way that they realize the benefits of SOA. It is based on the UML metamodel, thus being a MOF implementation. The IBM service model provides a level of abstraction that raises above the current implementation technologies, but does not directly represent any of them. This service model is based on a simple metamodel with a relatively small number of concepts, which should be reasonably familiar to the ones working with service oriented solutions.

The service model is realized as the UML2.0 Profile for Software Services [29] implemented in the IBM Rational Software Architect [15]. This profile is constructed with the purpose of providing a common language for describing services and views for different stakeholders, e.g. business analysts and software architects.

The model is not tightly bound to a given development process, but it is included in the Rational Unified Process (RUP) [30]. It is based on the ideas of Service Oriented Modeling and Architecture (SOMA) [31] and supports its life cycle activities.

Service Oriented Modeling and Architecture(SOMA)

SOMA is a service modeling technique used by IBM. According to [31], SOMA is the first method to realize an SOA.

It consist of three key steps : identification, specification and realization step. The identification step is focused on the identification of business processes in the domain. The specification step consist of designing the service specification, while the realization step is concerned of the architectural and design decisions.

Specification of service

The service is the center of attention in the IBM service model. It it is constructed to reflect the Web Service Description Language (WSDL) service. Even if conceptually the service is the centerpiece, from a modeling point of view it cannot exit independently. The model gives the priority to the *service specification* as well as the *service provider* for constructing a service that reflects the WSDL definition.

The model base its definition of service specification on a structural, a behavioural and a policy specification.

- The structural specification defines the operations that are called and the messages exchanged between these operations.
- The behavioural specification is used to describe the protocols or communication that takes place between services.
- The policy specification denotes the constraints on the services and policy assertions that may cover security manageability.

These definitions are equally important, but in some cases, depending on the type of service being modeled, they are optional.

4.1.2 UML for EDOC

The UML for Enterprise Distributed Object Computing (EDOC) [32] is a set of specifications that are used in modeling PIMs in MDA. It consists of :

- Enterprise Component Architecture (ECA)
- a metamodel and UML Profile for Java and EJB
- Flow Composition Model (FCM)
- a UML Profile for Patterns
- a UML Profile for ECA
- a UML Profile for MOF
- a UML Profile for Relationships

These specifications are used to describe architectures, business logics, behaviours and implementation of software systems.

UML profiles are used in designing EDOC systems. These profiles are specified in a MOF metamodel, independent from the UML metamodel. They tailor the language into specific areas [1], which in our case is the service domain. ECA forms the core of EDOC.

Enterprise Collaboration Architecture(ECA)

ECA [33] simplifies the development of component based EDOC through the use of a modeling framework, which conforms to the OMG MDA. It is a business component architecture that provides interoperable business components and services, promoting business orientation in modeling. The business component architecture combined with a collaboration-based modeling approach, aims to achieve a more loosely coupled integration [33].

ECA introduces Component Collaboration Architecture (CCA), which describes how to model the structure and the behaviour of the components that the system consists of. CCA is followed by a set of specifications defining entities, events and business process models. These models describe how to represent the concepts of the application domain, how to model the event driven systems and system behaviour in the context of the business IT support. Together they support the design of an EDOC system on the foundations of CCA. Their semantics are expressed in a MOF metamodel.

Application of ECA in EDOC systems

In this part we describe how the concepts defined by the ECA form a conceptual framework to design EDOC systems. The framework consists of five different viewpoint specifications, namely: enterprise, computational, information, engineering and technology specification. The combination of the models we described above, CCA, entity, event and business process model, together with the other specification of the UML profile for EDOC, ensures the development of the full set of these viewpoint specifications. Fig 4.1 shows the relations of the ECA concepts to the viewpoints.

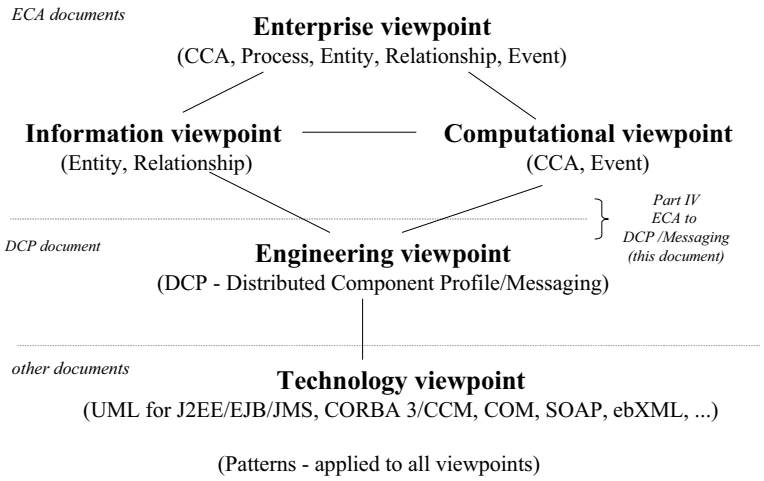

Figure 4.1: ECA elements related to the conceptual viewpoints

4.1.3 PIM4SOA

PIM4SOA [34] is an open-source project developed under the Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications Integrated Project (ATHENA IP) [35]. Based on the OMG MDA approach on platform independence and technology neutrality, this project aims to provide modeling tools and services for a SOA.

One of the results achieved by the PIM4SOA project is a multi-aspect metamodel, which goal is to define an abstract language that could be used to describe SOA in a platform independent way. The PIM4SOA metamodel is designed with the purpose of bridging the gap between system analysts and IT developers, resulting in the ability to support formal transition between enterprise models and system implementations [36]. in addition, this metamodel provides the platform neutral abstractions that can be used to integrate and define mappings to the main implementations of an SOA.

The PIM4SOA metamodel has been designed with the goal to keep the metamodel as simple as possible, but still providing the needed expressive power to model the

service domain. It is a Essential Meta Object Facility (EMOF)[1] implementation of concepts and ideas from general purpose approaches, like UML for EDOC, more specific metamodels like the Business Process Definition Metamodel (BPDM) RFP [37] and Web Service architecture, described in [5]. These concepts are formed in 4 different aspects that are described in the next section.

PIM4SOA metamodel aspects

The four aspects of PIM4SOA are the information, service, process and Quality of Service (QoS). The focus of this thesis is related to functional aspects of a PIM4SOA and we will discuss only the first three aspects of the metamodel, leaving the non-functional aspect, QoS, out of this description. The functional aspects that the metamodel identifies are:

Information is one of the most important elements of the enterprise that needs to be described. It is related to the structures or messages that are exchanged between services. This aspect is inspired by the EDOC specifications for UML, but uses mostly *Class* metamodel elements. The goal of this metamodel was to provide the needed expressive power to model information i SOAs, but with a simplified set of elements from the specifications named above.

Service represents the business accessible functionality. It is the abstraction and an encapsulation of the functionality provided. The metamodel is inspired by the web service architecture from World Wide Web Consortium (W3C), but in contrary to Web Services, the *service* in PIM4SOA is not explicitly defined in one concept. The service is defined as a **Collaboration**, which is a specification of interaction between participating roles. A **Service Provider** takes on this participating roles and realizes them in the collaboration.

Process describes the set of interactions between services and their sequencing of work. These interactions are described in terms of actions, flows, decisions and other elements. This aspect is inspired by BPDM, but with some modifications that allow integration with other business models and PIM4SOA elements. The process-aspect metamodel is closely connected to the service-aspect metamodel through the element that describes *behaviour*.

Figure 4.2 shows the identified metamodels for the respective aspects. In the bottom layer are placed possible PSMs. Artifacts of these models could be generated once a PIM4SOA model is defined [36]. In the ATHENA IP project, was realized a transformation of PIM4SOA to Web Services [34], while a transformation to BDI Agents is introduced in [38].

[1] subset of MOF

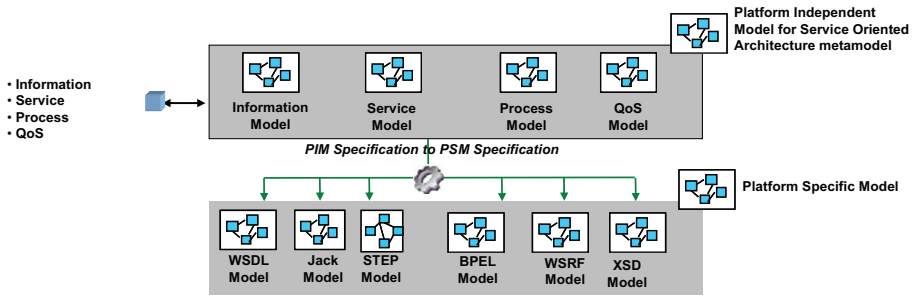

Figure 4.2: PIM4SOA metamodel overview

4.2 Editor frameworks

The following sections will explore 4 editor frameworks that are : Microsoft Visual Studio, Eclipse GMF, MetaEdit+ and XMF Mosaic. These frameworks are the state of the art in the aspect of editor generation and support approaches from different standards like Software Factories and Model Driven Architecture.

4.2.1 MS Visual Studio

This section concerns Software Factories and its DSL Tools, which are included in Visual Studio(VS) 2005. DSL Tools are a Standard Development Kit(SDK) that allows developers to use Microsoft's platform in building visual modeling tools [39].

Software Factories

> *"We suggest that the current software development paradigm, based on object orientation, may have reached the point of exhaustion, and we propose a model for its successor...."* Jack Greenfield [40]

The successor in this case is Software Factories, an old concept brought again to light as a Microsoft standard. Software Factories is a revolutionising way of building software, by industrialising the software industry [41]. It is a collection of ideas from MDD, Component Based Development (CBD) and software product lines. Using concepts as component assembly, domain specific assets or mass customization, Software Factories seeks to create a new application development model that will create cheap and quick development tools [40]. The realization of these ideas are now in progress and some of them appear in VS DSL Tools.

DSL Tools

The Microsoft DSL Tools is a suite of tools used to create, edit and visualize domain-specific data. The main purpose of DSL Tools is to construct custom designed tools

that can be used to model a problem domain [42]. DSL Tools provide a graphical designer and solution templates for constructing a specific language. It also contains code generation facilities that will generate a running designer based on a domain model and an XML description for the user interface.

The designer(editor) for the domain model is introduced in VS as the DSL Designer. It has built in the essential features for creating metamodels. The creation of the metamodel is not based on the MOF standard, but rather on the metadata framework Module Description File (MDF). DSL Tools also provide a wizard with a few solution templates for a DSL. Providing four different templates, the Designer Wizard aims to be a good starting point for a wide range of specific needs in graphical languages. These templates are resemblances of UML diagrams like class diagrams, activity diagrams or use cases. The developer can chose the solution template which is closest to the desired language and modify it, if necessary. A minimal language template is also included in case of the modeling language to be very different from any of the other solution templates.

To achieve the goal of creating a DSL Editor or as we called it before, custom designed tool, Visual Studio needs a way to define designer notations. In general, the designer notations involve shapes, connectors etc. In addition to the graphical designer for the domain model, DSL Tools provide an XML editor, which is used to define and edit the designer notations. These notations are then mapped onto the domain model, for example shapes to classes and connectors to relationships.

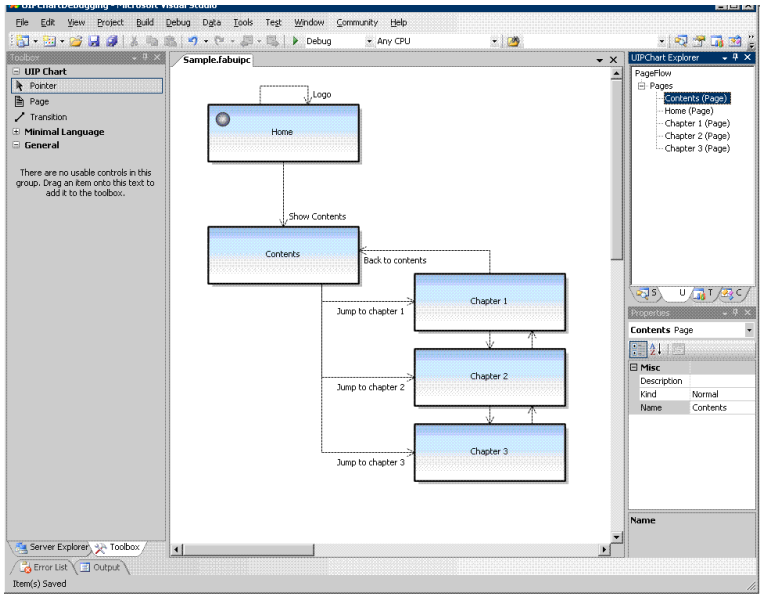

Figure 4.3: Generated editor in DSL tools

The custom designed tool(graphical editor) is created after the code for the designer is generated and the solution is built. The tool is a new instance of VS and can be used to create models based on the specified DSL. One of its main features is the

transformation of designed models into text(code). A customized designer is shown in Figure 4.3.

4.2.2 Eclipse GMF

> *"The Eclipse Graphical modeling Framework(GMF) provides generative component and runtime infrastructure for developing graphical editors. "* [43]

The GMF project is an open-source project that started to help people create graphical editors in Eclipse [44]. The main idea is to reduce the complexity in creating the editors with this frameworks and also to come near to the MDD approach by testing some of its concepts. The graphical editors are based on both EMF [45] and GEF [46] . [2]

GMF architecture

EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. It is a Java [47] open source framework that aims to improve the design and implementation of structured models. The aim is realized by assuring model persistence and the support of code generating from different type of models, like Ecore, Xml Schemas etc [48].

GEF on the other hand is also a modeling framework, but it is concerned with the graphical representation of elements. It allows the representation of any kind of models. The graphical representations are done with a standard 2 dimensional drawing framework called Draw2D [49]. This provides GEF with the ability to easily develop graphical representation for nearly every model [48].

GMF is an extension of GEF and EMF, providing a bridge between these two technologies. This framework is constituted of two main components: runtime and generation(tooling). The runtime component binds together EMF and GEF, particularly the command infrastructure which provides a number of diagramming components that can be reused in the editor [50]. It also provides additional features like extended metamodeling facilities or transaction support.

The generation consist of a handful of tools that gives the developer the opportunity to define in detail the specifications for the generator. It introduces some specific metamodels that are instantiated at the editor's runtime or during its generation. Below we explain the models that are used in the process of generating the editor.

Editor generation

In order to generate the editor, we need to define the concept that will be represented in it and their graphical visualisations. The concept are provided from an input EMF

[2] In the implementation of MED4SOA we used the GMF Callisto version 1.0.1 incorporated in the Eclipse Platform 3.2.1

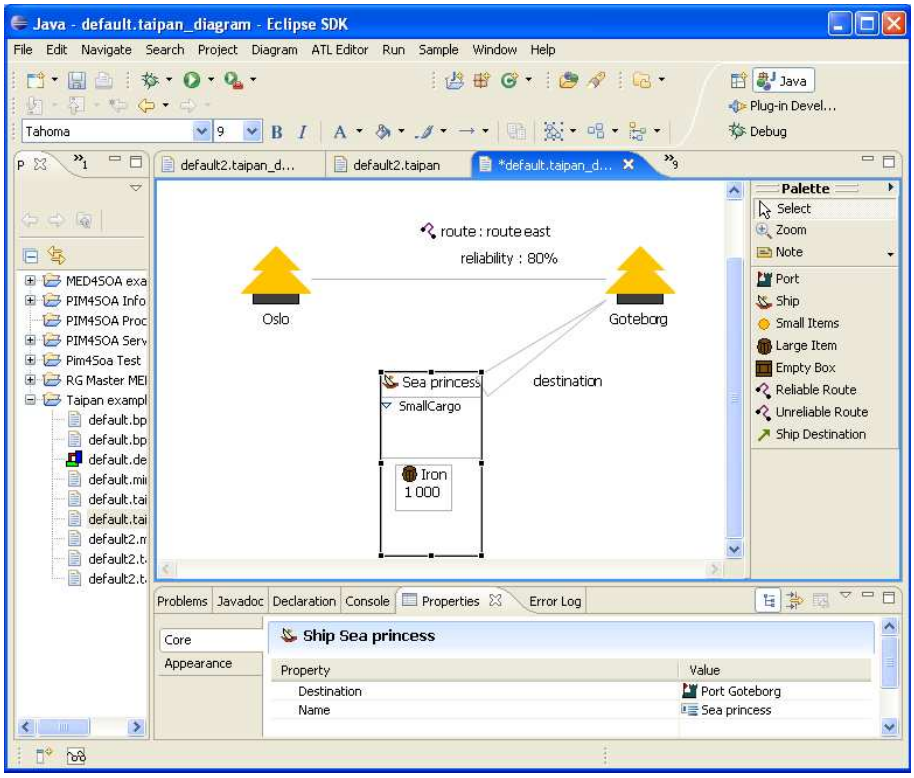

Figure 4.4: A graphical editor created in Eclipse GMF

metamodel which defines also their semantics, while the graphical visualisations are defined in three models :

The graphical definition model gathers information about the graphical elements that will be represented in the editor. It describes compositions of GEF figures like: node, connection, compartment, label etc.

The tooling definition model gives a definition of the editor palette and other additional main menu items.

The mapping definition model creates the coupling between the graphical elements and their underlying semantics that reside in the metamodel. A mapping to a tooling device is also created.

The mapping definition model is then used to create a generator model which includes the necessary information for the code generator. On its turn, the code generator will create the diagram code, which is the actual implementation of the editor. GMF editors run as a new Eclipse application. Figure 4.4 shows a generated editor in Eclipse GMF.

4.2.3 MetaEdit+

> *MetaEdit+ is an integrated environment that allows building of modeling tools and generators fitting to a specific application domain, without having to write a single line of code* [51].

MetaEdit+ is a multi-user and multi-platform environment that supports system development and method development at the same time. It is a realization of MetaCase technology, which promotes customization of methods to the application domain and to the user's need.

MetaCase

MetaCase technology offers the ability to easily capture the specifications of any method. Computer Aided Software Engineering (CASE) tools are then automatically generated based on these method specifications. In contrast to other CASE tools that are constructed based on a specified method, MetaCase technology offers a CASE tool that could update the method in accordance with changes in the application domain or development environment. This is made possible by its three layers architecture [52].

While traditional CASE tools rely on a two-level architecture, namely a programmed metamodel[3] and the compiled tool, MetaCase tools are based on a three-level architecture. They include the traditional two levels, plus a metamodeling language level which is placed on top of the metamodel. This level allows the developer to modify the metamodel, which results in reducing the limitations and making the tool more flexible.

MetaEdit+ : A CASE of MetaCase

MetaEdit+ is the CASE tool that implements the MetaCase technology. The tool serves to construct a DSL as a metamodel with all the domain specifications, behaviours and constraints, and also specify the mappings to code, in a code generator [53]. For this method implementation, MetaEdit+ is equipped with a metamodeling language and tool suite for defining the method elements. The tool suite has generic CASE behaviour for objects and relationships, including Diagram Editors, property views and other object browsers. The developer needs only to create the language, with the object types and the definition of the relationships between them. The desired symbol is given to the element, using existing symbols or new created on the symbol editor. In this way there is no need for hand coding [54].

In addition to the above functionality, MetaCase+ includes a code generator based on a pre-specified DSL. The code generator allows the developer to modify the DSL, providing more flexibility in going through the model, and generating the desired

[3] usually it is only the tool vendor which has access and can modify the metamodel

code. MetaCase+ also provides XML import/export and an API for data and control access to the tool's functions. Figure 4.5 illustrates a diagram editor created with a purpose of modeling a family tree.

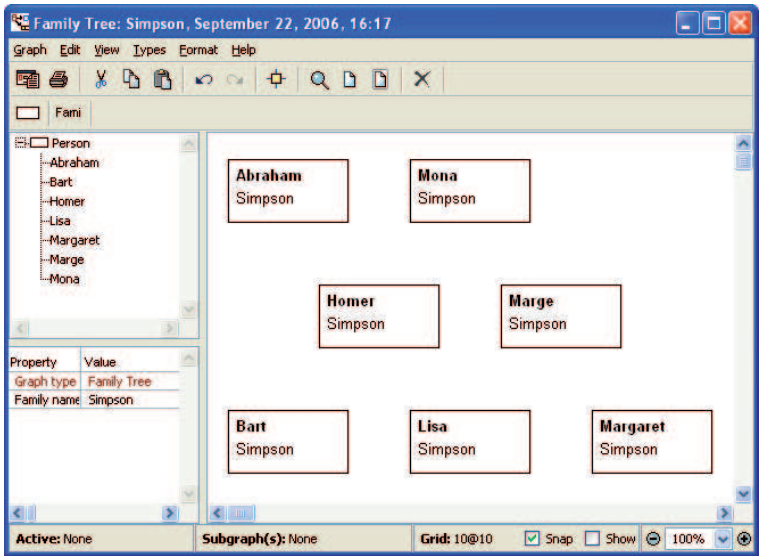

Figure 4.5: Illustration of MetaEdit+

4.2.4 XMF Mosaic

XMF-Mosaic is a model-based development platform created by Xactium Limited. The platform is created with the intentions to reduce the enterprise application development costs and at the same time to improve the quality and agility of the development process [55]. In the need of change in the way software are built, Xactium Limited brought the solution to these issues with the LDD approach, realised in XMF-Mosaic. With this platform, developers have no longer limited possibilities to represent the problem domain. They can develop languages, processes and tools tailored especially for the domain.

XMF

XMF is an abbreviation of (eXecutable Metamodeling Facility). This facility is an extension of the existing standards such as MOF, QVT or OCL, with executable metamodeling capabilities. It provides mechanisms that can capture the concrete syntax, abstract syntax and the semantics of a language. In addition it supports creation, editing and execution of platform independent metamodels [17].

XMF-Mosaic is the development platform that implements XMF. Based on Eclipse and XMF, with a key feature, which it is completely modelled in itself, this platform aims to provide firstly a rich environment for language design and execution. Secondly it

provides a richer metamodeling facility, in particular, in terms of languages build for the purpose of mapping between models. Thirdly and most importantly, it provides the ability to generate editors and use these tools to manipulate the concepts of the domain.

XMF-Mosaic features

XMF-Mosaic presents as one of its main features the capability to create domain models using a rich XMF modeling tool. This domain models can then be extended with constrains and operations. They are fully executable using the eXtensible Object Command Language (XOCL), which is an extension of OCL.

As mentioned above the key feature of XMF-Mosaic is its completely self modeled characteristic. This is made possible by XCore(the kernel), a collection of self-described classes that gives the platform more extensibility. In this way the developer can extend the existing metamodels or concrete syntax definitions.

Another important feature is model to model transformation. XMF-Mosaic includes two types of mapping languages. The first one, called XMap is a pattern oriented mapping language expressing unidirectional mapping. The second is called XSynch and is used in bi-directional synchronisations between models.

Generation of editors

XMF-Mosaic uses a collection of languages, called XTools, to construct a diagram tool for a metamodel. The diagram tool, or custom editor, is an instance of XMF-Mosaic with a user interface used to create, manipulate and execute domain concepts. It is

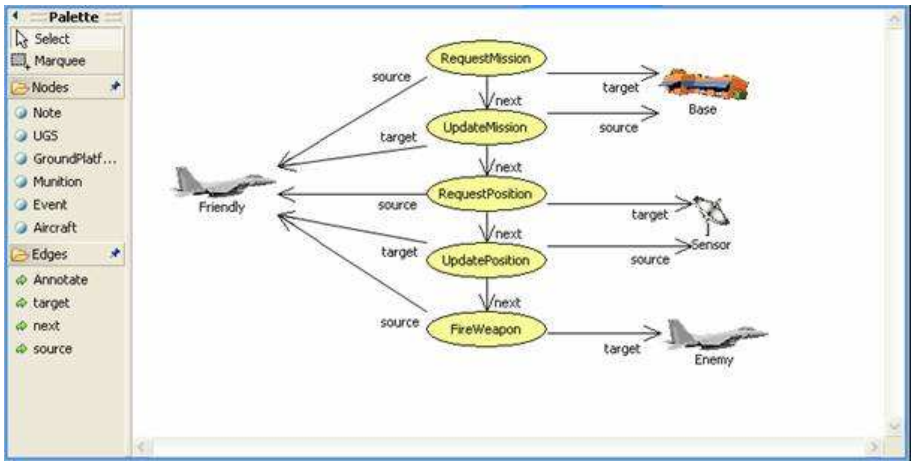

Figure 4.6: Generated editor in XMF Mosaic

created by deploying the metamodel of a domain to an XTool. This is done automatically from an early prototype generator included in the platform or specified manually through textual definitions.

XMF-Mosaic uses a textual language, implemented through the XTools classes to construct the tool components. The two most important components are the tool type and the element manager. The tool type component allows construction of different types of tools(diagrammatic, form etc), while the element manager takes care of the event handlers. These definitions are then compiled and the user-interface created and displayed. Figure 4.6 shows a generated editor in XMF-Mosaic.

4.3 Choice of technologies

This section provides an evaluation of the technologies described in the previous sections. This evaluation is done on the basis of the requirements listed in section 3.4 and 3.5, namely the requirements for a SOA metamodel and the requirements for a graphical modeling editor.

It is important to note here that the intent of this evaluation is to find the most suitable metamodel and editor framework for *our* solution, MED4SOA. We do not intend to suggest a metamodel to the UPMS RFP nor to find the perfect editor framework. The evaluation of the metamodel is done through theoretical research and analytical argumentation, while the frameworks are evaluated both on individual practice and analytical argumentation.

We evaluate the proposed technology by giving them a score[4] from 0-2. The maximum of 2 points is assigned when a requirement is fully met, 1 point is assigned when the requirement is partially met and 0 points are assigned when a requirement is poorly met. The points are then added to form the score.

In the next subsections we provide the evaluation scores first for the metamodels and then for the editor frameworks.

4.3.1 Metamodels

Table 4.1 provides an evaluation overview for metamodels presented in section 4.2. The scores are followed by a short argumentation for the results of each of the metamodels.

IBM service model

MR1 The IBM service model is an extension of the UML metamodel, which is constructed from MOF.

[4]non-empirical evidence

	Requirement	IBM service model	UML for EDOC	PIM4SOA
MR1	MOF-based	2	2	2
MR2	Platform Independent	1	2	2
MR3	Business-oriented	1	2	1
MR4	Loose coupling	1	2	1
MR5	Expressiveness	1	1	1
MR6	Extensibility	2	1	2
MR7	Simplicity	1	0	2
	Score	9	10	11

Table 4.1: Evaluation overview for SOA metamodels.

MR2 Even though it is promoted as raised above the current implementation platform and has all the characteristics of a PIM, this model is built with the intention to reflect the WSDL service description and not other implementation technologies.

MR3 It does not define concepts that represent information structures of an enterprise.

MR4 It has the ability to provide the necessary service abstraction, even though business behaviour is not incorporated.

MR5 It provides means for representing the service specifications, but has its short coming in the concepts of interaction between services, explicitly service behaviour.

MR6 Due to its roots on the UML metamodel, and taking under consideration that UML is a general purpose modeling language, this model has the ability to allow additional specifications.

MR7 The relevant small number of concepts that the model consists of, makes it very simple to comprehend, but lacks the ability to express business process information.

UML for EDOC

MR1 The specifications of UML for EDOC are also an extension of the UML metamodel, thus MOF-based.

MR2 Its main goal is to provide a PIM in MDA and it fully meets this requirement.

MR3 Through ECA, introduced as a business component architecture, it provides the needed concepts of describing business oriented elements.

MR4 It has the ability to represent the service as a fully self-contained block that incorporates the business behaviour.

MR5 These specifications of UML for EDOC do not give a explicit definition of the service, nor to its description.

MR6 In the same way as the IBM service model, it has the ability to allow additional specifications, but with a large number of specifications its potential to expand even further is not high.

MR7 UML for EDOC with all its specifications makes a relatively large and complicated model. Recalling its five viewpoint specifications, going from the enterprise to the technology level, we can conclude that it does not promote simplicity.

PIM4SOA

MR1 PIM4SOA is a metamodel created from the facilities of EMOF, a subset of MOF.

MR2 It is a platform independent model, and the mapping to more than one specific platform proves its independence.

MR3 It lacks in the description of the hole enterprise, but still has the ability to include the fundamentals for describing the information and behaviour.

MR4 Its concept of service is not limited to any standard or technology, but the representation of the service as a collaboration of roles and a service provider has its limits.

MR5 PIM4SOA through its three fundamental aspects has the ability to provide most of the concepts of the service domain, but lacks the ability to give a full service description.

MR6 The basic element of PIM4SOA relies on UML specifications, so it has the full potential to extend in several aspects. created with the specific purpose, namely to describe the concepts of the service domain.

MR7 It awards simplicity, due to the relatively small concepts that are included in the metamodel, while still being able to describe the main concepts of the service domain.

4.3.2 Editor frameworks

In this section we evaluate the editor frameworks based on the requirements listed in section 3.5. Table 4.2 gives an overview of the scores and we proceed by arguing shortly for the results of each of the editor frameworks.

MS Visual Studio

GFR1 MS Visual Studio is a product of the Microsoft corporation. Hence it fails to meet the first requirement.

GFR2 It is a fully metamodel-based tool even if the metamodel is not created from MOF.

	Requirements	MS VS	GMF	MetaEdit+	XMF Mosaic
GFR1	Open Source technology	0	2	0	0
GFR2	Metamodel-based	2	2	2	2
GFR3	Flexibility	1	1	1	1
GFR4	Low implementation effort	1	2	2	1
GFR5	Isolation of notations	2	2	2	2
GFR6	Extensibility	1	2	1	1
GFR7	Multiple view support	2	2	2	2
	Score	9	13	10	9

Table 4.2: Evaluation overview for graphical editor frameworks.

GFR3 Its DSL Tools have the ability to represent the domain concepts. It lack the flexibility in representing the elements graphically. The graphical representations for the time being are limited to predefined nodes and edges.

GFR4 This framework has a relatively medium implementation effort.

GFR5 It gives a clear isolation of notations, through the domain model and designer.

GFR6 Since it is a Visual Studio implementation it supports very few extensions.

GFR7 The visual interface of the editor supports both graphical and property view of the elements.

Eclipse GMF

GFR1 Eclipse GMF is an open-source project.

GFR2 It is created to build editors based on the metamodel and checks for correctness against the metamodel.

GFR3 Even though elements can be visualised in figure structures and icons, complex figures require additional implementation with GEF.

GFR4 GMF provides a low implementation effort due to its generative approach.

GFR5 It ensures the isolation through the graph and tool model.

GFR6 The editor is designed with all extensibility features provided by the Eclipse architecture and the underlying EMF and GEF technologies.

GFR7 It supports visualization of elements graphically, tree like and through a property view.

Meta Edit+

GFR1 MetaEdit+ is a commercial tool of MetaCase.

GFR2 One of its main features resides in the fact that it is fully metamodel based.

GFR3 Its flexibility lacks also in the complexity aspect of the representations.

GFR4 The implementation effort is low due to its features based on CASE tools.

GFR5 It has the ability to isolate the concepts of the metamodel from the graphical representation due to its generative structure.

GFR6 Even though it provides the option of extending the metamodel of MetaEdit+, this task is very complex and requires additional effort.

GFR7 MetaEdit+ with its CASE tools can visualize the elements in different ways, from diagram editors, to property viewers and other object browsers.

XMF Mosaic

GFR1 XMF Mosaic is also a commercial framework promoted by Xactium Ltd.

GFR2 The framework provides tool support for LDD being also an initiative of Xactium Ltd.

GFR3 Similar to the other candidates it has the possibility to represent most of the elements graphically, but fails to fully represent complex elements.

GFR4 Complexity is present also in the implementation, thus raising the implementation effort.

GFR5 It has a clear distinction of the notations in both generator and textual definition of the graphical elements.

GFR6 Since it is fully modeled in itself this framework promotes extensibility, although complexity is a big factor in this context.

GFR7 The framework is based on Eclipse and supports the visualization of elements graphically or in a property viewer.

4.4 Summary

In this chapter we explored candidate metamodels and generative editor frameworks for MED4SOA. In the end we concluded with an evaluation that pointed out the most suitable metamodel and framework editor.

The scores from Table 4.1 indicates PIM4SOA to be the most suitable metamodel for MED4SOA. This metamodel will provide the concepts and abstractions that will be graphically realized in the editor.

The scores from Table 4.2 gives the consent for the Eclipse GMF technology to be the editor framework in which we will implement our solution.

Chapter 5

MED4SOA architecture and design

"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."

Douglas Noel Adams

THE architecture and design chapter will focus on the specifications needed to construct MED4SOA. It describes its architecture, concentrating on the components that constitute this graphical editor. We will go in details on the main parts of MED4SOA, PIM4SOA and GMF, trying to give the a clear view of the metamodel and the graphical framework. The last part of the chapter will give a description of the development process and the generation overview.

5.1 Architectural overview

In chapter 4 we investigated different metamodels concerning on the service domain, and technologies that could realise graphical editors for these metamodels. Based on the scores these technologies received, we concluded that our solution will be build using Eclipse GMF technology and it will realize the concepts of PIM4SOA. The realization from the GMF aspect, consists of a graphical visualisation of the elements specified in the metamodel. In this way we can create a model that is an instance of PIM4SOA metamodel.

From a very abstract architectural view point, MED4SOA is the product of putting together PIM4SOA and GMF. Figure 5.1 gives an overview of that view point. We can identify in the figure the three aspects, of the PIM4SOA metamodel, being the information, service and process aspect. In the upper-left side of the figure is shown a coarse overview of the elements that GMF consists of. MED4SOA has the same underlying technology of GMF, with an additional generated layer that forms the graphical editor.

Recalling from section 4.2.2, we introduced GMF consisting of two parts: generation and runtime. The generation part is used in constructing the graphical editor, while

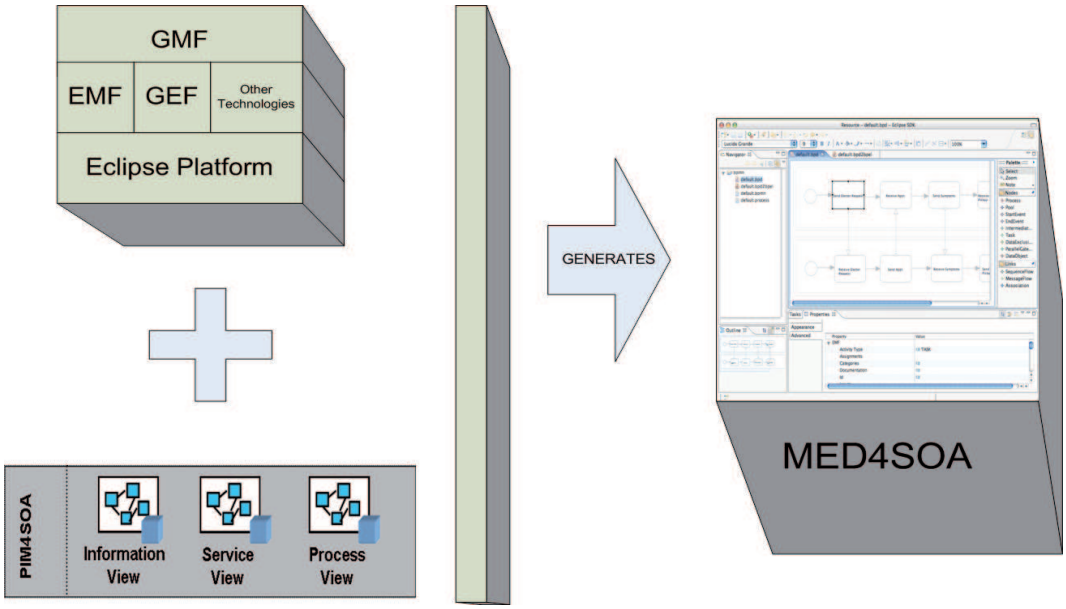

Figure 5.1: Overview of MED4SOA elements

the runtime is the bridge between EMF and GEF. Together with the underlying features of the Eclipse Platform, GMF gives MED4SOA the needed functionality to run a graphical editor. Figure 5.2 shows these elements and the connection between them.

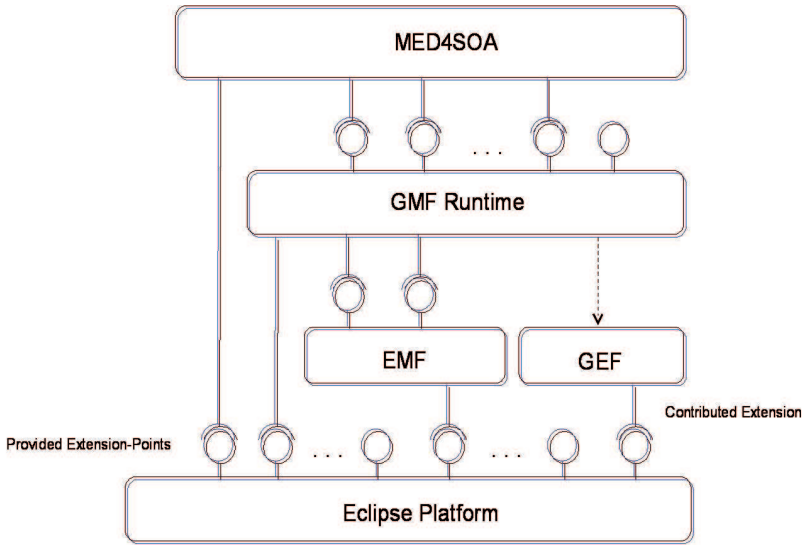

Figure 5.2: Overview of MED4SOA architecture

In the previous figure, the elements are connected to each other through extension

points[1]. These are interfaces provided in Eclipse to promote extensibility. We call the technology added to the extension point as a contributed extension. Worth noting from the figure, is the fact that GMF does not technically extend GEF. It is an extension of EMF features, but in the same time using the functionality of GEF. This explains why GMF is called a bridge between EMF and GEF.

Below we continue our exploration of the PIM4SOA metamodel and the technologies forming Eclipse GMF. In difference to chapter 4 we will concentrate in more technical details, aiming to provide a more specified view of the technologies involved in the construction of MED4SOA.

5.2 PIM4SOA specifications

In section 4.1.3 we argued for PIM4SOA to be a metamodel which goal is to describe the abstractions of the service domain. In this section we will have a closer look at its specifications. The purpose is to give a clear view of what the specifications define and how they are connected to each other. A complete description of PIM4SOA elements is given in the ATHENA IP deliverable A6 [56], while a tree-like visualization of the metamodel, in EMF style, is provided as Appendix A. In the tree-like visualization we can see the details that could not be included in the representation of PIM4SOA as three different diagrams [2].

In order to adapt the metamodel to GMF, we mad small modifications to PIM4SOA. These modification did not change the purpose of the metamodel aspects. They only changed some characteristics of the elements, especially the containment features. GMF does not support a domain element to be contained by more than one domain element. It results in a inconsistence storing of semantic data. The changes undertaken affected the service and process aspect.

We start the explanation of the three aspects of PIM4SOA with the information aspect and continue further with respectively the service and process aspect.

5.2.1 Information-aspect metamodel

The information aspect is used to model what is exchanged between services in terms of documents or messages. It is highly inspired by the UML for EDOC model, and resembles to the UML 2.0 *Class* metamodel, but has a reduced set of concepts. A graphical diagram of the metamodel is shown in Figure 5.3.

The *Document* is the centerpiece of the information-aspect metamodel. This element is a generalization of a *Package*, which is a UML 2.0 metamodel element. It represents a specific structure that may include other elements. Usually a document will contain structures of information that in PIM4SOA are described as *Entities*.

[1]The number of the extension points in the Eclipse platform or the layers above is purely fictional. These extensions are used only to give an idea of the technology used and not the accurate number of extension points used to implement it

[2]One diagram for each metamodel aspect

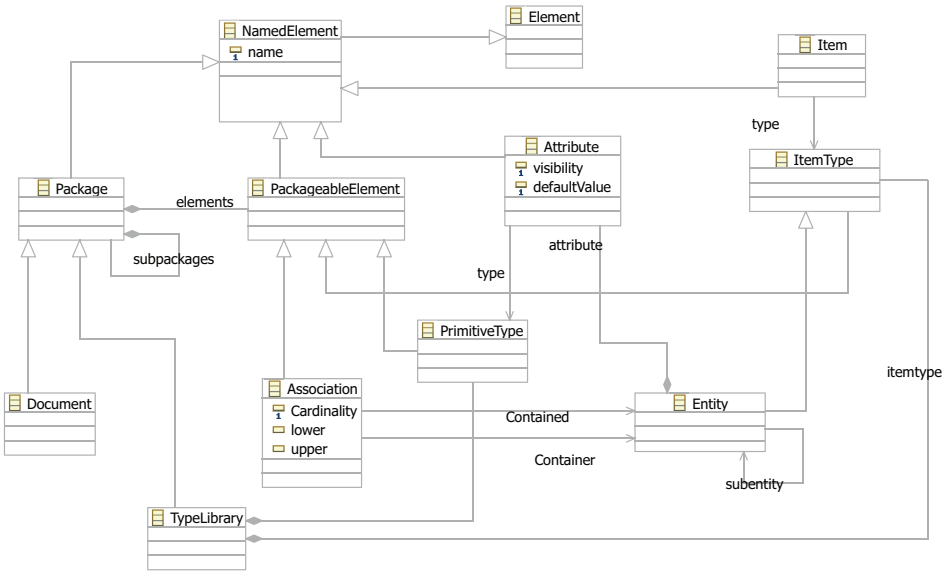

Figure 5.3: Information-aspect metamodel

Entities are elements used to describe complex structures of information. They resemble to a UML 2.0 *Class* and similar to this standard element, they have *Attribute*s, or can engage in *Association*s with each other. An entity is a generalization of an *ItemType* in PIM4SOA.

ItemTypes are the building blocks of information in this aspect of the metamodel. They can represent UML or Java primitive types like String, Integer but are also used to create self defined types. ItemTypes are collected in a *TypeLibrary*, which like Documents is also a package.

Another element of this aspect should also be the *Message* element, but due to its close connection with the element *Role* it is included in the service-aspect metamodel.

5.2.2 Service-aspect metamodel

The service-aspect metamodel is created with the goal to model functional structures from one or more systems that want to share their capabilities with others. It goes very close to the definition of *service* from OASIS Reference Model [18], but in the same time inspiration is found in the Web Services architecture. Figure 5.4 gives an overview of the service-aspect metamodel.

Even though the service aspect is inspired by the web service architecture, PIM4SOA does not have a clear definition of the service. In order to provide abstractions that could model *service*s from one or more different systems, this metamodel suggests that a service should be modeled as a *Collaboration*. The element Collaboration specifies patterns of interaction between the *Role*s participating in this interaction.

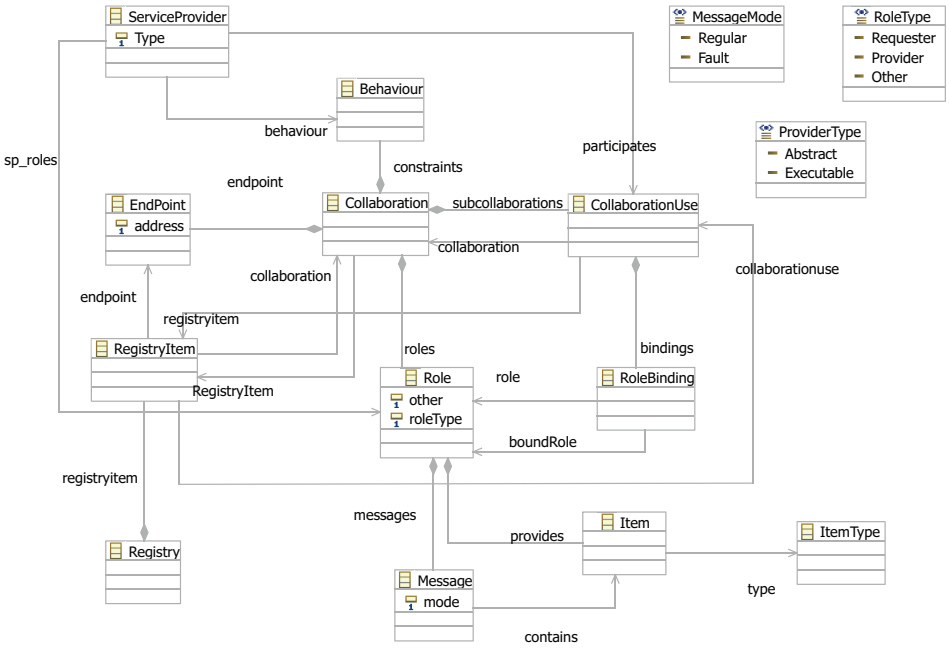

Figure 5.4: Service-aspect metamodel

A *Collaboration* and the *Role*s participating in it, give a vague abstraction of a service. The service-aspect metamodel fills the picture[3] by introducing the *ServiceProvider* and *CollaborationUse*. The first element takes on a participating role and realizes it in a collaboration. The second is the glue that puts together a Collaboration and a service provider. It literally describes the usage of a collaboration and how the service provider participates in it. The collaboration use specifies also the relation of a role to a service. This relation is represented as a *RoleBinding*.

The changes to the service aspect were mostly made on the service provider. In the metamodel developed in the ATHENA project, the behaviour, role and collaboration use concepts are aggregated to the service provider [56]. We changed these aggregations to simple association, in order to have an element contained in only one other element. The containment features were also changed in the role element. Our version of PIM4SOA specifies that an item is aggregated to a Role, but only associated with the message.

Other items worth noting in this aspect are the *Registry* where the *Registry Item*s are specified. A registry item contains information about the collaboration, collaboration use and *EndPoints*. The end points represent an identifying address for the service. Another important element is the *Behaviour* of the service provider and collaboration, which specifies a message sequence in the service. More on behaviour and process modeling is to come in the next aspect of PIM4SOA.

[3] the concept of a service as described by OASIS

5.2.3 Process-aspect metamodel

The process-aspect metamodel is found on concepts submitted in the OMG's BPDM RFP [37]. It aims to provide abstractions that will allow modeling of interactions between services. These interactions will include the operations and flows that occur in the process. Figure 5.5 gives an overview of the elements included in this aspect of the metamodel.

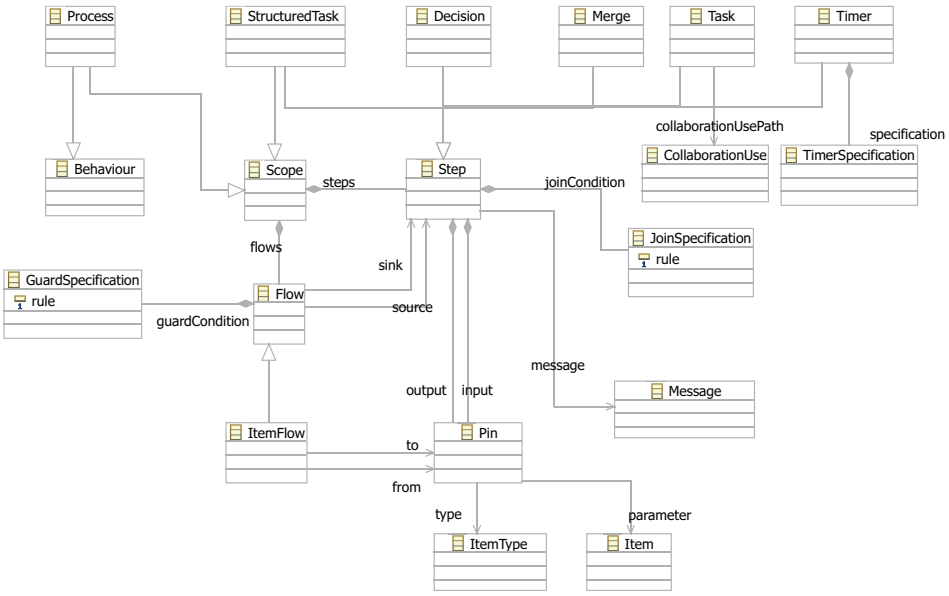

Figure 5.5: Process-aspect metamodel

The service-aspect is closely related to the service-aspect metamodel through the element representing behaviour. A *Behaviour* is the abstract class that connects the service to the process, by marking the *Process* element as a generalization of both a *Behaviour* and a *Scope*. This allows a scope to be instantiated as a process that belongs to a service provider or a collaboration.

A process consists of *Step*s and *Flow*s. A *Step* is defined in this aspect of PIM4SOA as a single node in a process. It is represented as a *Decision*, *Merge* or most of the time as a *Task*.

Tasks are the building blocks of a process. They represent the actions in the process, but can also be calls to other service providers or more specialised operations. These specialised operations will require further implementation that needs platform specific data. Tasks are linked to each other through *Flow*s.

The specification of flows is quite complex, but is created with the intention to provide the modeler with much flexibility. The flows can be simple *ControlFlow*s, or connected to the information part of the metamodel as *ItemFlows*. To reduce the complexity of the flow between steps we decided to drop the *Interaction* element that was added to

provide input and output interfaces to a step. Instead we connect the flow directly to a step, and leave the definition of interfaces to the service aspect of PIM4SOA.

In the case of information flow between tasks, *Pin*s are added in this element. These abstractions define an input or output for a specific item type. A pin includes also a parameter that is represented as an item or set of items required in the interaction.

The three aspects of the metamodel described above form the basics for MED4SOA. The concepts that these aspects describe form the abstract syntax of MED4SOA. As mentioned in section 2.3, in order to form a complete solution we need to specify the concrete syntax. In the next section we will look at the technology that will make possible the realization of the concrete syntax, namely the visualization of the concepts of PIM4SOA with GMF.

5.3 MED4SOA architectural elements

The Eclipse GMF project was introduced in section 4.2.2 and we described in bold lines the structure of GMF and its architecture. In this section we will explore in more details, the technologies that GMF builds on top of, respectively EMF and GEF. We will explain how these technologies together with the new GMF features provide the infrastructure and components for developing visual design and modeling surfaces in Eclipse. We start from the bottom, by saying a few words about the Eclipse platform.

5.3.1 Eclipse platform

Eclipse is an universal platform, created by the Eclipse Foundation, in order to provide an open development environment comprised of extensible frameworks, tools and runtime for building, deploying and managing software across its life-cycle [44]. The Eclipse platform is known for its tool integration, utilizing an open and extensible architecture based on plug-ins. Figure 5.6 gives an overview of the Eclipse Platform architecture.

Eclipse has a small runtime kernel and a collection of plug-ins that are activated as needed. This kind of architecture provides all functionalities through the actual plug-ins. They are implementations or as we called them before, contributions to extension points and connect to the eclipse core through an interface, called an extension point. We introduced GMF, EMF and GEF as plug-ins in Figure 5.1, while in Figure 5.6 we can notice two other important elements of the Eclipse platform that are Java Development Tools (JDT) and Plug-in Development Environment (PDE). These contribute in facilitating Java programming and plug-in development respectively.

5.3.2 Eclipse Modeling Framework (EMF)

Earlier in the chapter we described PIM4SOA and concluded that it forms the basics of MED4SOA. All the concepts of the metamodel described will form the input for

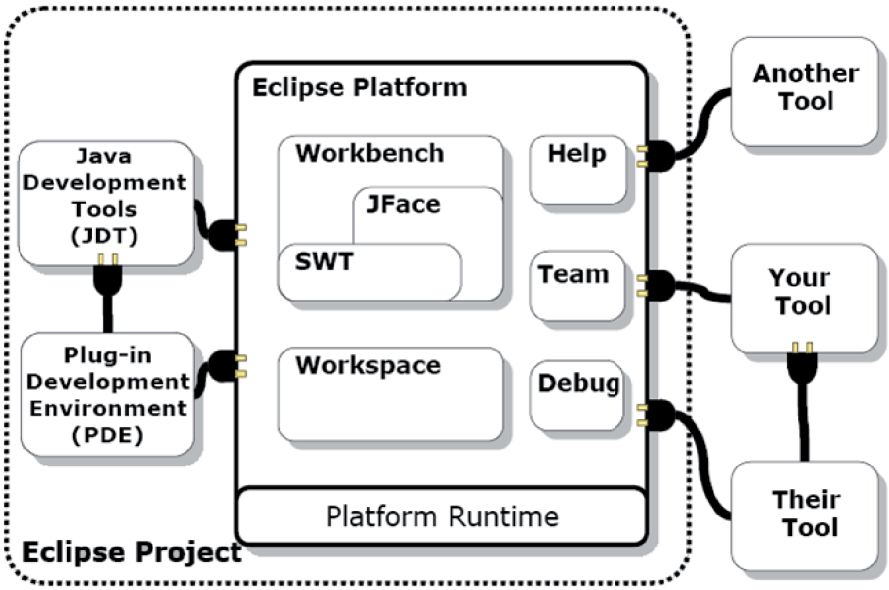

Figure 5.6: High-level architecture view of the Eclipse platform

the graphical editor, but as such, they need to be defined in a form understandable by GMF. This is made possible by EMF.

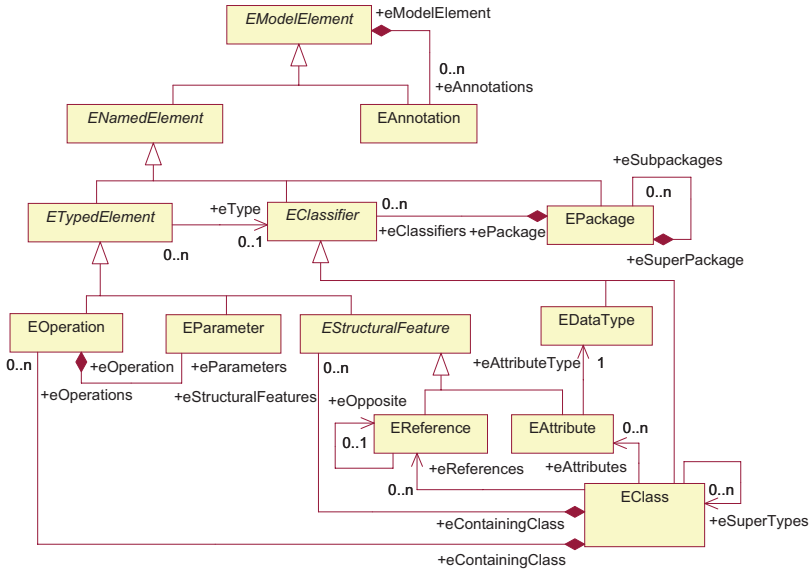

Figure 5.7: Kernel of Ecore model

EMF is a modeling framework for describing class models and generating Java code. Here we introduced two different aspects of EMF. The first one, modeling framework for describing class models resides in the fact that EMF provides a core model for creating metamodels. The metamodels in MOF are class-like models. EMF can construct similar metamodels that are based on the subset of MOF, EMOF. The constructions, uses EMF's kernel, which is called the Ecore model. Fig 5.7 shows the main part of the Ecore model. The kernel model contains elements like EClasses, EAttribute or EReference that define respectively classes, attributes and relations. The PIM4SOA metamodel is constructed using these elements and graphical diagrams of its aspects are shown in Figure 5.3, 5.4, 5.5. Physically the created metamodel is an XMI file.

Except the ability to create the metamodels, EMF allows the generation of a set of Java classes from the constructed metamodel. These Java classes will provide the model persistence. They are the basic support for further processing of the instances of the metamodel, including operations like creating, deleting, loading or saving. Furthermore these Java classes support the manipulation of the elements in a basic tree-like editor, which will not be needed in MED4SOA. Instead, the Java code will be used to support the manipulation of these element in the generated GMF editor.

Next step in our exploration of the elements composing MED4SOA is to take a look at the technology that realizes the visualisation of the abstractions implemented in a EMF model.

5.3.3 Graphical Modeling Framework (GEF)

The Graphical Eclipse Framework is an Eclipse project practiced in building graphical editors. This framework provides the developer with two important components in building a graphical editor. The first one is a lightweight plug-in, called **Draw2d** that will provide the graphical components. The second is an Model-View-Controller (MVC) framework built on top of Draw2d, which adds editing capabilities to GEF.

Draw2d

Draw2d is a toolkit of graphical elements, called *figures*, which are simply Java objects. The toolkit has the needed ability to associate these figures to a Standard Widget Toolkit (SWT) Canvas [57] and assign listeners to them. Listeners are elements that dispatch any event to an event manager or update manager. These managers are included to support efficient layout of the figures. With layout we mean the process of finding out the location of the figure in the drawing.

Other features included in Draw2d are painting of the figures, connection layouts, flexibility in the coordinate system, tool palette support and also user actions like undo, redo, delete and so on.

GEF MVC

While Draw2d creates the figures and their layout, the purpose of the GEF MVC framework is to facilitate the display of these figures by supporting interactions from mouse or keyboard. The MVC paradigm is an architectural pattern that aims to separate the data represented in the model from the user interface represented by the view [58].

The MVC pattern in GEF applies also the same rules. A controller is the only connection between the view and the model. The controller in GEF is a structure called *EditPart* and there exits an instance of the controller for each visualized object. In MED4SOA the controller will be the link between the EMF model and all the graphical elements that will be shown in the editor. Figure 5.8 illustrates the GEF MVC architectural pattern.

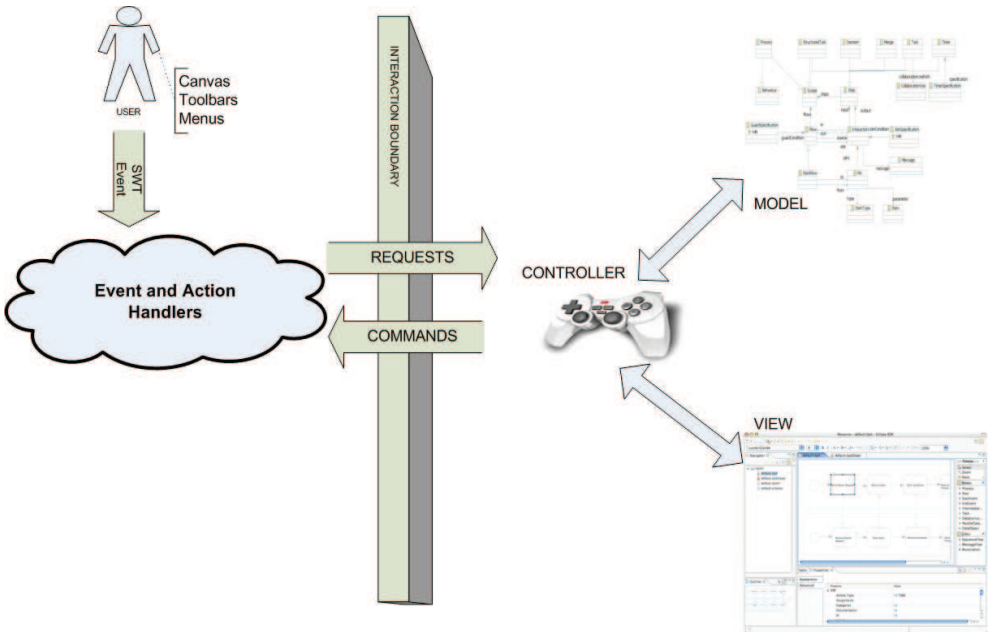

Figure 5.8: Illustration of GEF MVC architecture

We mentioned above that GEF is used to construct graphical editors. The process of creating a graphical editor with GEF, is manual and fairly complex. We would need to hard code the figures, controllers, event handlers and other components involved in the development. Instead, we will use the functionality provided in GMF, which provides the ability to connect the abstractions of the metamodel to pre-constructed GEF figures through automatically mapping. Below we explain the features of GMF.

5.3.4 GMF features

The GMF project is created with the aim of building a bridge between EMF and GEF. While it is an extension of the features of EMF, it uses the abilities of GEF to facilitate the building of graphical editors that are based on EMF models. Utilizing GMF, the development of the graphical editor gains an initial set of graphical elements already implemented in GEF, and together with the much improved interoperability between GEF and EMF, leads to a reduced implementation effort.

GMF is equipped with a set of metamodels used in both its main parts, runtime and generation. The runtime uses a notation metamodel which is the main link between GEF and GMF. The generation aspect utilizes also a number of specific metamodels, which define the models used during the design of the visualization elements and in the code generation for the new graphical editor. Below we explain the models introduced in GMF. For more information on these models and a complete picture of the GMF architecture, check the developers guide in [59].

The notation metamodel

The notation metamodel is one of the new metamodels introduced by GMF. Its purpose is to store the visual information of running diagrams like locations, positions or other features concerning nodes or edges. Since it is an EMF metamodel, the persistence abilities of EMF are used in the storage of information. The notation metamodel is completely domain independent and the only connection to the underlying semantic model relies in the *View* element. A graphical representation of the notation metamodel is given in Figure 5.9.

The *View* element is the centerpiece of this metamodel. It is the element that defines shapes and is divided in respectively three kind of shapes, *Diagram*, *Node*s and *Edge*s. These form the main components of a Diagram. The tricky thing in the metamodel is that the diagram element itself contains nodes and edges.

The lower part of the diagram shows elements that include information about nodes and edges. Nodes in a diagram will have their *LayoutConstraints* as *Ratio*, *Size* and *Position*, while edges are constrained through *Endpoints* and *Anchor*s. The enclosure of GEF information[4], in the persistent model shows clearly the bridge between EMF and GEF.

The graphical definition model

The graphical definition model is used to define the visual elements that will be shown in the graphical editor. These visual elements are the nodes, connections, compartments and labels that will form the diagrams in the editor. Together with the ability to define these elements, the graphical definition model defines a figure gallery with reusable, pre-constructed GEF figures. These figures will be the representations of

[4]endpoints, anchors or layout constraints

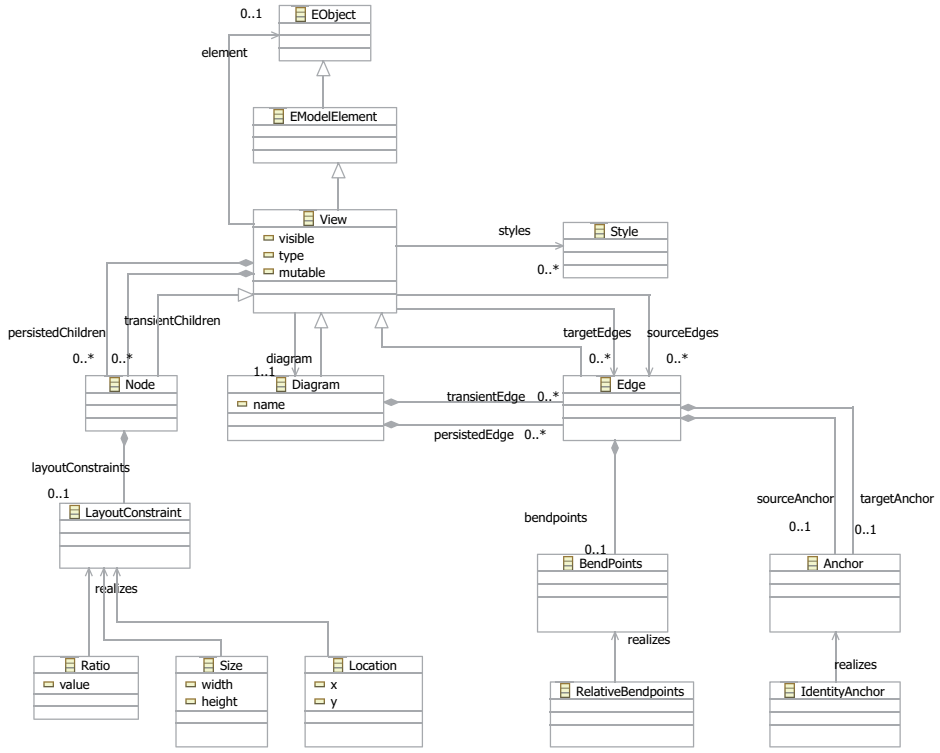

Figure 5.9: Graphical representation of the GMF notation metamodel

the nodes, edges and their labels. In the same time the graphical model allows construction of other composite GEF figure, but it requires ulterior knowledge of figure construction in GEF.

The graphical definition model is not a runtime element. It is only used during the construction of the graphical editor and the information included is fed to the code generator, which in will generate the figures[5]. Usually the elements of these models are constructed to represent the abstractions of the domain model, but the graphical definition does not have any connection (yet) to the domain abstraction. It only provides a wizard, which gives the developer a kick start with defined diagramming elements for each abstraction in the metamodel.

The tooling definition

The tooling definition model is included in GMF to cover the aspect of palettes, menu elements, toolbars and other actions associated with these. In MED4SOA the tooling definition will be mostly used to create the palette with the *Creation Tools* for each and every element that will be shown in the diagram. We will use standard elements that are included in the GMF runtime for menus, toolbars and pop-ups.

[5]recalling that the figures are Java classes

The palette in MED4SOA will be mostly divided into two parts. The first section will include creation tools for nodes and labels, while the second will have the creation tools for the connections. Note that compartments are incorporated in the structure of a node and they do not need a creation tool.

GMF uses standard icons for the creation tools that are shown in the palette. We will customize the palette with our own icons. The icons are tiny figures, which will also be shown on the diagram element.

The mapping definition model

At the introduction of the graphical definition model we mentioned that the graphical elements do not yet have any connection to the underlying semantics, described in the metamodel. The connection between the graphical definitions of the diagram elements, with their domain abstractions and the respective creation tool is defined in the mapping definition model.

The mapping definition model creates different kinds of mapping. It can start by *CanvasMapping* which is the general mapping of a diagram surface to the domain model and the palette. Further down, the mapping goes on *NodeMappings* and *LinkMappings*. They will map respectively each node and connection to the element in the domain model. The creation tools for the mapped elements are also included in the defined mappings.

LabelMappings and CompartmentMappings have different kinds of structures. These mappings can not stand alone and are owned by NodeMappings and LinkMappings. The LabelMapping is used to visualise attributes of the elements in the metamodel to the respective diagram element that represent it, while the CompartmentMapping creates a logical grouping of nodes in a compartment figure defined in the graphical model. The grouping which is the composition of nodes into another, is realized by introducing the notation of ChildReference. In this way the mapping definition model creates a hierarchy of parent-child elements in the diagram.

In the mapping definition model GMF provides also mechanisms that will constrain the model. These constraints are used in collaboration with EMF and Eclipse Modeling Framework Technologies (EMFT), which incubates other technologies extending or complementing EMF [60]. Constraints are used in nodes, links or in an *Audit container* that will validate the model created by the editor in the runtime.

The generator model

The Graphical, Tooling and Mapping definition models, as indicated by their names, are utilized to define the graphical elements and their associations to the semantics in the domain model. They create the full picture of the future graphical editor, but anyhow these definitions do not have the generation-specific data that are required in order to generate the Java code [59]. The specific details for the generation are defined in the generator model.

The generator model is created in order to simplify the code generation by creating a model that lies between the high-level definition models described above and the generated Java code. This model is created by transformation from the other models and will include information for implementation specifications. The code is generated by using Java Emitter Templates (JET) [61].

Most of the diagram code is generated without further modification to the generator model. Sometimes, depending on the developer and the generation target, small modifications are made. These modifications include changing names, id or similar data.

So far we gave an overview of MED4SOA's architecture, and explored its main elements, namely the PIM4SOA metamodel and GMF with its underlying structures. In the next section we will have a look at the process of developing MED4SOA as a group of GMF projects.

5.4 MED4SOA development process

The realization of MED4SOA starts with the creation of a GMF project and finishes with the generation of Java code that will be used as the diagram plug-in. Figure 5.10 shows an overview of the stages of the development of MED4SOA in a kind-of activity diagram style. To simplify our work we decided to create a GMF project for every aspect of the PIM4SOA metamodel. In the figure below the names of the models created, except the metamodel, are all *med4soa* followed by the extension of the model file. In this way we generalize for the three aspects of the metamodels and avoid to repeat them in the description of the process.

The next step in the process is to develop the domain model, graphical and tooling definition model. These three models do not necessary need to be created in the order of introduction. As mentioned above the development of the graphical definition model can be boosted by the starting wizard where the diagram elements are created. It can also be developed independently from the domain model, just to define figures that will be visualised in an editor. Similarly, it also applies for the tooling definition model. In the development of MED4SOA we applied both approaches.

The PIM4SOA metamodel is not a complicated metamodel, but the number of elements is not small. Considering this, we created the main diagram elements using the wizard and processed further by adding other elements manually. The same approach was applied also in the tooling definition model. The results of this stage were two models, *med4soa.gmfgraph* and *med4soa.gmftool*, describing respectively the graphical and the tool definition model.

The domain model in MED4SOA will be an EMF model of PIM4SOA. The metamodel is in a typical EMF format, Ecore, and named *pim4soa.ecore*. PIM4SOA was created before GMF and an EMF tree-diagram editor was used to construct the metamodel. With the introduction of GMF, new features aroused. One of those is a graphical ecore diagram, used mostly to create metamodels in a class-like diagram editor.

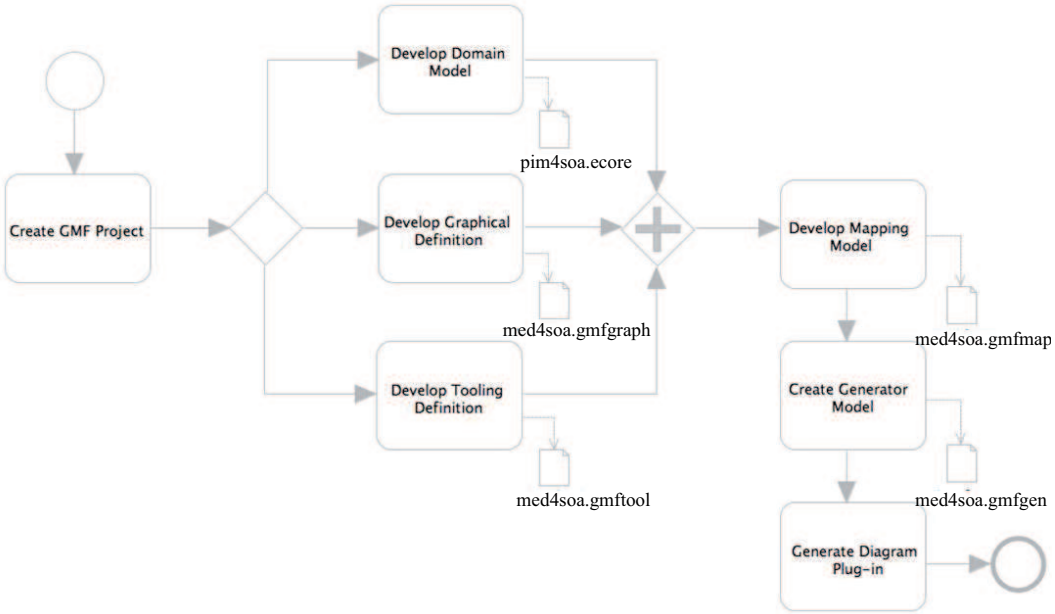

Figure 5.10: The process of creating MED4SOA

The mapping definition model is created after all the domain, graphical and tooling specifications are in place. It will map these specifications to each other and form a complete picture of the future graphical editor. The product of this stage is the model named *med4soa.gmfmap*.

The generation model is the last model in the development process. The model, *med4soa.gmfgen*, will be created with input from the previous models and supply the necessary data to the code generation.

The last stage in the process is the code generation. For each GMF project involved in the development, a diagram plug-in is generated. The plug-in will contain the graphical editor implementation code. This code can be run directly as a new Eclipse instance or included in the plug-in directory of the Eclipse platform.

5.5 Summary

In the previous sections of this chapter we described the architecture and design of MED4SOA. We explored the metamodel it is based on, giving a description of the definition in all its three aspects. Further, we explained the technologies that GMF builds in top of, Eclipse, EMF, GEF. We described their functions and how they are connected to MED4SOA.

About GMF, we gave a detailed look of its two main aspects: runtime and generation. Respectively, the notation model, graphical definition model, tooling definition

model, mapping definition model and generator model were described. In the end, we concluded with the description of MED4SOA development process.

Here we conclude the architecture and design chapter and leave the description of the realization and implementation details to the next chapter.

Chapter 6

MED4SOA realisation and implementation

"A great many people have come up to me and asked me how I manage to get so much work done and still keep looking so dissipated. My answer is 'Don't you wish you knew?'"

Robert Charles Benchley

So far in the development of MED4SOA we chose the metamodel that will be realized in the editor, the technology that will implement it and lately we described in details its architecture. In the following chapter we will concentrate in the implementation of our solution. The chapter will cover different aspects of the implementation process. It gives a description of PIM4SOA's implementation. Further it will explain some details in the implementation of the views(aspects) in the editor.

We decided to cover most of the details of the implementation in the *information* aspect of MED4SOA. The decision was made due to the low complexity of this aspect and the similarities in the implementation details with the other two. In this way we will explain the work done in implementing the information view of the editor and only cover additional implementation details for the two other views.

The result of MED4SOA's implementation will be 9 different eclipse projects that will perform the role of the plug-in. Three projects, namely **org.athena.pim4soa, org.athena.pim4soa.edit** and **org.athena.pim4soa.editor** contain the implementation of the metamodel, while the rest implements the three different views of the editor. Their name will be an extension to the first project, for instance the service view will have the definition models implemented in the **org.athena.pim4soa.graphical.service** and the diagram code will be generated in **org.athena.pim4soa.graphical.service.diagram**. These plug-ins are available at [34][1].

Before starting the explanation of the realisation and implementation details, we want to remind the user that even though the research method used in this thesis has in

[1]The project could be downloaded from the SVN repository, under v3

it descriptive elements, the rest of this chapter is not meant to be a GMF tutorial. Our intentions in writing this chapter were to explain the work done in implementing MED4SOA. For further GMF implementation details, please consult different tutorials in [43].

6.1 Implementation of the metamodel

The PIM4SOA metamodel is realized, applying the facilities of EMF. In the description of EMF, section 5.3.2, we mentioned that EMF provides a core model practiced in the creation of metamodels. The metamodel is created using the elements of the Kernel Ecore model shown in figure 5.7. In a EMF style the metamodel would be implemented using a tree-editor, but instead we used the new features included in GMF.

6.1.1 Ecore diagram

With the introduction of GMF, the tree-editor was extended, resulting in a diagram editor with the purpose of creating metamodels. We used the diagram editor to create our metamodel, constructed as *pim4soa.ecore*. The diagram editor applies the same elements of the Ecore model and visualises them graphically in forms of nodes and links. This editor is created from the same GMF mechanisms that we will use in the development of MED4SOA, and is included in the GMF release as an *Example Ecore Diagram*. Figure 6.1 shows the different aspects of PIM4SOA metamodel implemented in an Ecore diagram. The figure shows a diagram of PIM4SOA, a view of the metamodel and that is why it has a different file name.

We created the different aspects of PIM4SOA using *packages* implemented in ecore as an *EPackage*. These packages contain the elements that form the metamodel. The implementation constitutes of *EClasses* for the concepts of the metamodel and *EReferences* for the relationships between them. The concepts could also have attributes, which are realized by *EAttribute*.

6.1.2 EMF generator model

When the metamodel is in place, we continue the realization of MED4SOA by creating an EMF model. This model is named *pim4soa.genmodel* and is used to specify information concerning the generation of the code. It will specify information that is not included in the ecore model, like the generation target or plug-in IDs.

The generated code has different purposes and is divided in model, edit and editor code. The model code is the representation of the metamodel elements in Java classes. The edit code is a set of reusable Java classes, which will provide editing features to the model classes and including a command framework for a future editor. This command structure was built initially to support an EMF editor, editor code, but instead it will be used by the diagram code of the GMF editor, which we will generate.

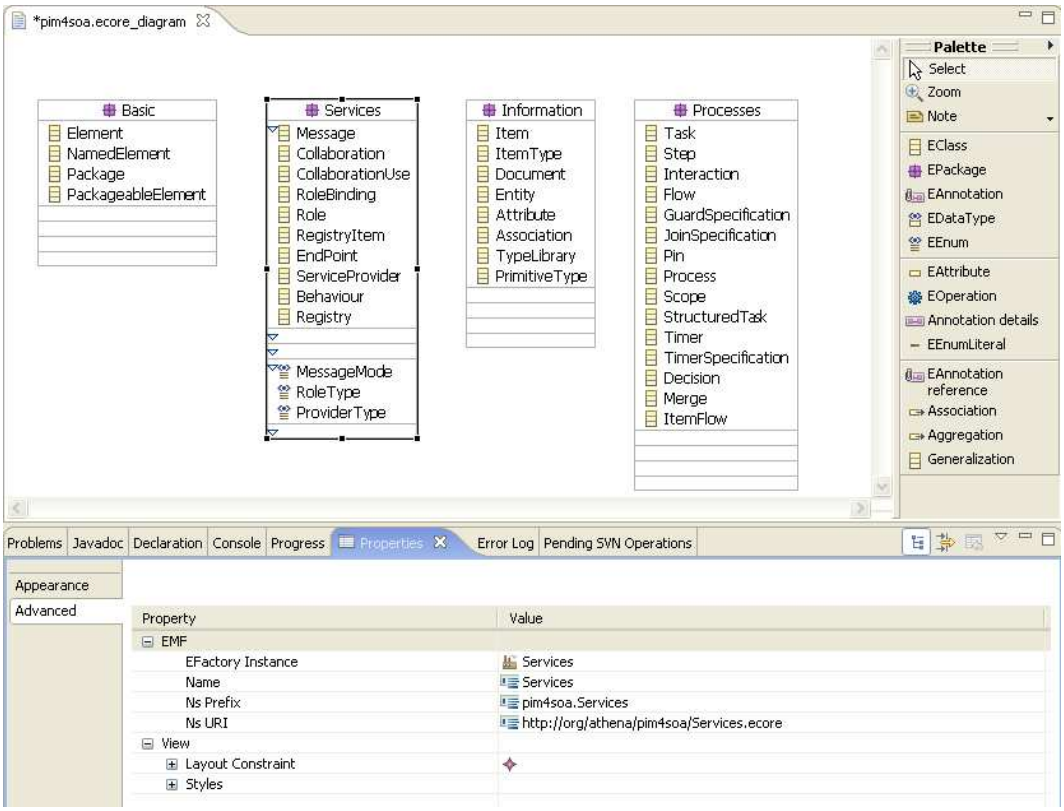

Figure 6.1: Overview of PIM4SOA in an Ecore diagram

6.2 Information aspect

This part of the chapter will describe how the definitions of information aspect of PIM4SOA are implemented in MED4SOA. Reminding once again that these definitions will be represented as visual diagram elements in the editors to come, we will explain to the implementation details of MED4SOA in this aspect. The set of diagram element will form the model for the information aspect of MED4SOA.

We start by exploring the graphical definition model, and continue further in exploring the tooling definition model, the mapping definition model. In the end we conclude with the generator model of the information aspect.

6.2.1 Information graphical definition model

The information graphical definition, as mentioned in section 4.2.2 and 6.3.4, will define the diagram elements that forms the future editor. These definition are included in a *Canvas*. In the information part of MED4SOA the canvas is named *Canvas MED4SOA Information*. The graphical definition model of MED4SOA is shown in Figure 6.2 as a tree-like editor.

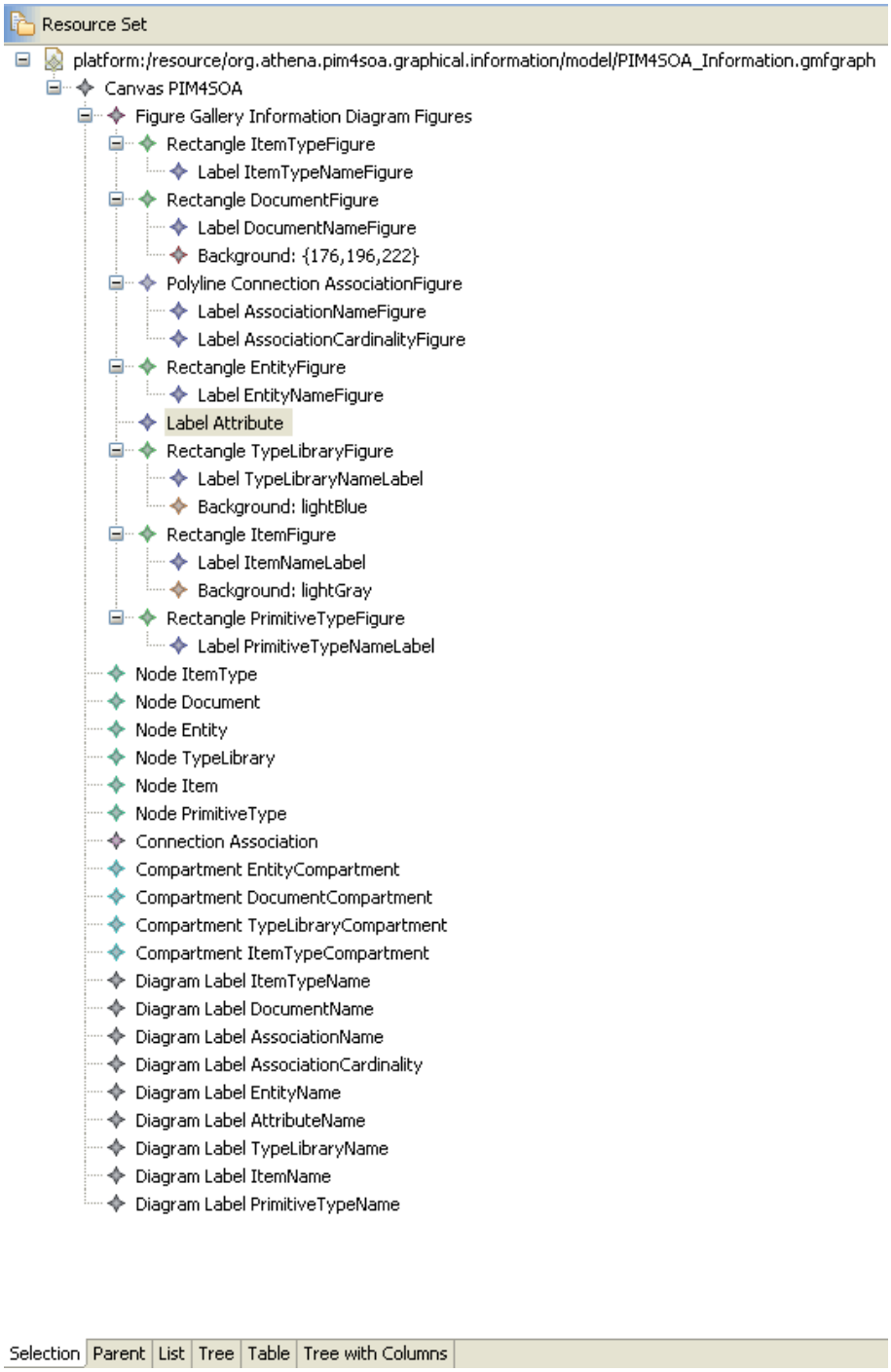

Figure 6.2: The Graphical Definition Model of the information aspect

The Canvas contains two kinds of elements. On the upper part are included the figures, being preconstructed GEF figures contained in a figure gallery. Beneath the fig-

ure gallery are listed the diagram elements that will be visualised by these figures. The diagram elements are represented by nodes, connections, labels and compartment. For each one of these elements, except the compartment, a figure is created. The compartment does not have an own figure, because it belongs to a node.

Most of the created figures are rectangles for the nodes, and polylines for the connections. These figures are customized through the property view, where we alter their properties or link them to other figures. Figure 6.3 shows the property view for a polyline that will represent an *Association*. In the property view named the polyline and link it to another figure. The other figure is also a polyline, created to represent the target of the association as an arrow head.

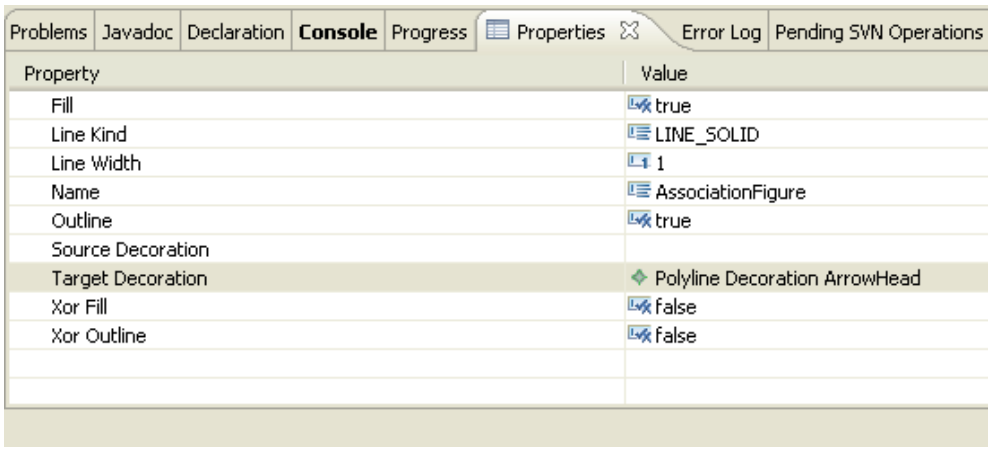

Figure 6.3: Property view of the association figure

The figures can include other elements which are called *children*. A fine representative of a child in a figure is a label, created in order to visualize the attributes of the metamodel elements. Other representatives can be distinct figurative properties like background or foreground, layout constraints, borders or even other figures. The inclusion of figures into others creates composite figures in the diagram.

The graphical definition model, Figure 6.2, does not show the relation of figures to the diagram elements. The connection is defined in the property view of the diagram element, by specifying which of the figures in the figure gallery will represent the element. An example of how a *Document* will be visualised in MED4SOA, is given in Figure 6.4. In this property view we specify that a Node Document will be visualised as a rectangle named DocumentFigure. Other specification in the property view are the node's name and the resize constraints.

6.2.2 Information tooling definition model

The tooling definition model is less complex than the graphical definition model. In this model we only define the creation tool for each and every one of the diagram elements. There exists also other options that will customize the menus and pop-ups,

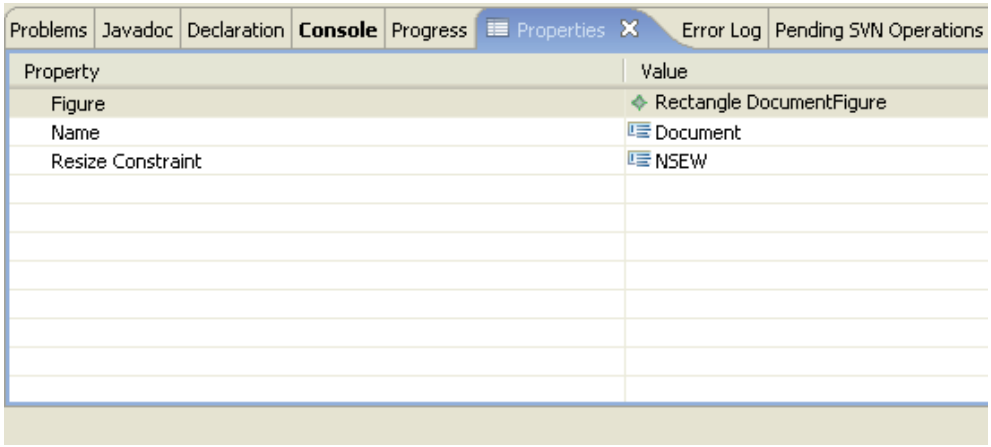

Figure 6.4: Property view of the document node

but we concentrated our work in the representation of the actual diagram elements and used the standard menus and pop-ups provided by GMF. The tooling definition model of the information aspect is provided in Figure 6.5

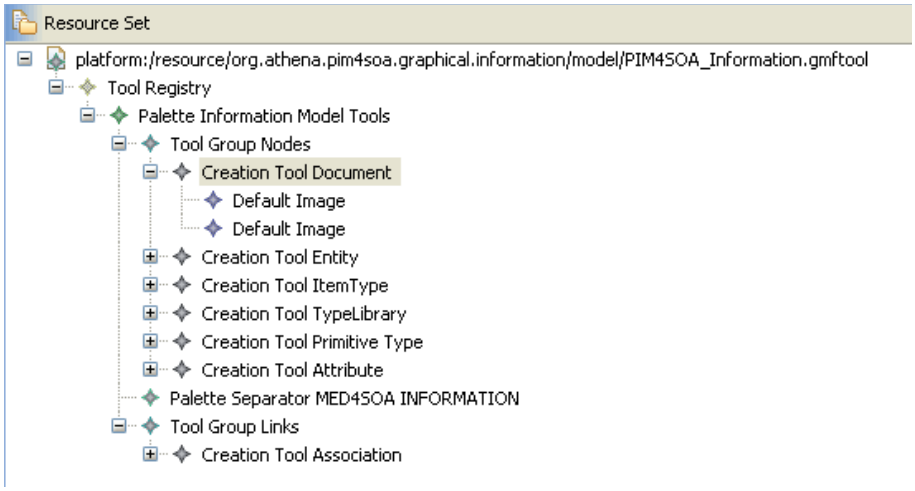

Figure 6.5: The tooling definition model of the information aspect

In the model we created a palette that will consist of the creation tools for the diagram elements. Included in the palette are also other standard GMF creation tools like *Note* or *Note Attachments*. The creation tools are given the name of the diagram element which they respectively will create. In the figure we can see the creation tool for the nodes like, *Document, Entity, ItemType, TypeLibrary or PrimitiveType*, labels like the *Attribute* and the connections represented by an *Association*. We divide nodes from the links by a palette separator [2].

[2]Attributes are labels, being neither nodes nor edges. They belong to a node, hence they are included

The creation tools in the palette consist of two other *children*. These children will define the icons used in the palette tools and the element created by them. We use the default icons from GMF, but those default icons we replace with our own. The replacement takes place in the generated **edit** package of metamodel implementation. In such a way the icons are used by either the GMF editor or an eventual EMF editor, if desired.

6.2.3 Information mapping definition model

The information mapping definition model will create the connections between the concepts of the metamodel, the diagram elements we created in the graphical definition model and the creation tools for these elements, which are defined in the tool definition model. In the same way as in the previous definition models, the mapping model is edited in a tree-like editor. Figure 6.6 shows the mapping definitions for the information aspect of MED4SOA.

The mapping starts with a *Canvas Mapping* that is located in the bottom of the model. The *Canvas Mapping* is created automatically by the start up wizard. In the wizard the developer is asked to identify the source models(domain, graphical and tooling). These sources are loaded in the model and they are shown in the previous figure below the canvas. Here we can see the metamodel as *pim4soa.ecore*, the graphical definition as *MED4SOA_Information.gmfgraph* and the tooling definition as the *MED4SOA_Information.gmftool*.

Canvas Mapping

The canvas mapping uses the property view for further editing. The property view of the canvas mapping in the information aspect of MED4SOA is shown in Figure 6.7. The view is divided into three parts. In the upper part, called the *Domain meta information* we define the domain model to be the *EPackage* pim4soa and the element used as the root diagram element will be a *Package*. The root diagram element points out the element that will contain the rest of the other elements created in the future editor.

The middle part of the view is used to specify connections to different part of the tooling definition. As mentioned above, in the tooling definition section, we used only the creation tool features and not the other definition elements. That is why in the property view we only define the palette to be the *Information Model Tools* and leave the menu or toolbar contributions blank.

The third part of the view identifies the visual representation. There is only one field in this part, and it is the *Diagram Canvas*..We set this field to be the canvas created in the graphical definition model, that is *Canvas MED4SOA Information*.

in the node section

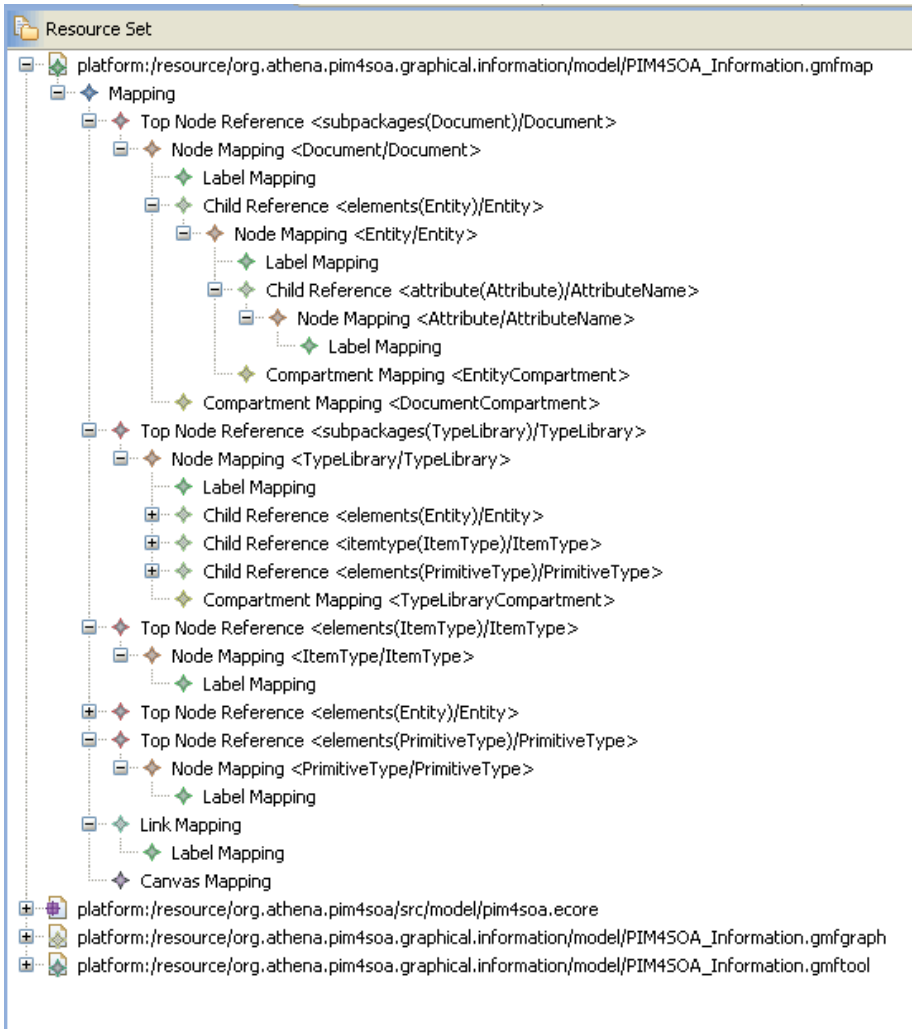

Figure 6.6: The mapping definition model of the information aspect

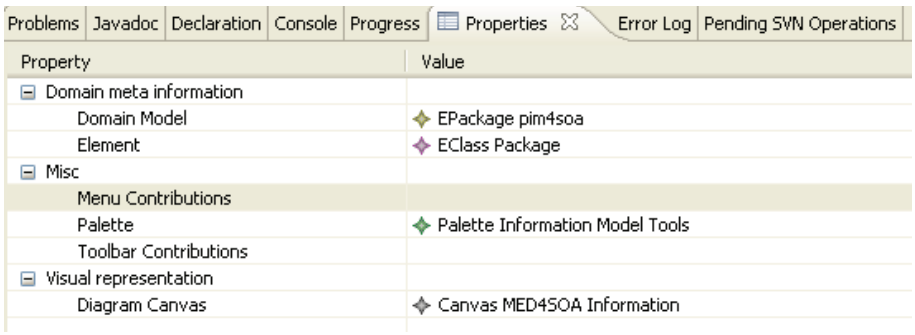

Figure 6.7: Property view of a canvas mapping in the information aspect

Node mapping

In order to map a node to its respective element, a node mapping is created. This kind of mapping together with the rest of the mappings that will be explained below in this section, could be created automatically by the wizard. Rather we chose to do this work manually by using the tree model that was shown in Figure 6.6. The manual approach followed gave us a better overview of the mappings that were needed.

The editing of a node mapping happens in a usual GMF style, through the property view. The Figure 6.8 shows the property view of the node mapping for the element Document in the information aspect of MED4SOA. In this view we can see the diagram node for a *Document* mapped to our *Node Document* from the graphical definition, with its corresponding domain definition mapped to *EClass Document* and the creation tool specified to be the *Creation Tool Document*.

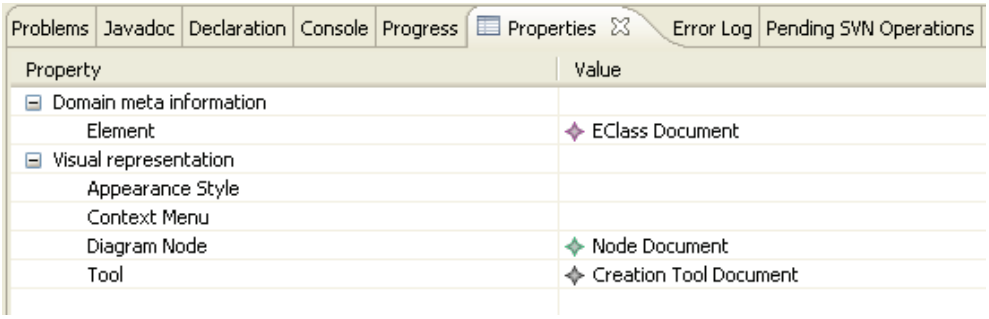

Figure 6.8: Property view of the node mapping for a document

A node mapping is contained in a GMF mechanism called a *Top Node Reference*. In here we specify the containment features of the element that will be mapped. In the case of a *Document*, we set the containment feature to be a *subpackage*. This means that when we create new diagram elements of *Document*, its instances will be added to the *subpackage* reference of our diagram root, that is a *Package* [3]. The instance is represented by its name, being the result of a label mapping that we will described below.

Label mapping

A label mapping will create the connection between a diagram label, to its graphical representation and the related domain element which is mostly an *attribute*. Mostly diagram elements have a names that identifies them. The name will be initiated together with the element, hence a creation tool for a label is not needed. The Figure 6.9 shows the property view of the label mapping included in the node mapping of the diagram element *Document*.

[3] In the metamodel we defined a package to include other subpackages

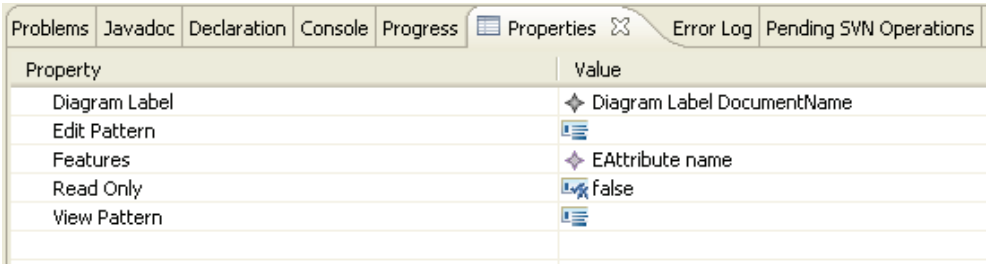

Figure 6.9: Property view of the label mapping for the attribute element

In the property view we specify that a label, intended to be the name of the diagram element *Document*, will be represented by the diagram label *DiagramName* and is a mapping to the *EAtribute name*. Recalling the information aspect of the PIM4SOA metamodel the element Document does not have an attribute *name* but we defined a *Document* to be a *Package* which is a generalization of a *NamedElement*. It is in here we find the attribute *name*. Except the *Compartement Mapping*, the rest of the mappings in this definition model will have a label mapping for the name.

Compartment mapping

A compartment mapping is a straight mapping of a graphical element to a diagram element. Given that the graphical element of a compartment is a part of a node graphical element, we only need to include the compartment mapping into a node mapping and specify its visual representation. In the case of the diagram element *Document*, its compartment will be represented by the *DocumentCompartment*. The compartment allows the diagram element *Document* to be composition of other diagram elements, which are defined in a *Child Reference*.

Child Reference

A Child Reference is used to include other diagram elements into a node. The children will be placed in a compartment specified for the node. Figure 6.10 shows the property view of the a Child Reference of the document Node Mapping. *Entities* will be the referred children.

A *Child Reference* is very similar to the *Top Node Reference* with the only difference that it will contain a *Node Mapping*, which is included in another *Node Mapping*. In the figure above, note that the containment feature for this child reference is specified to be an *element*. The reason for this, is the fact that the node mapping in the child reference will map an *Entity* to its respective representation and elements. Recalling again from the metamodel, an *Entity(PackageableElement)* is referred as an *element* from a *Document(Package)*.

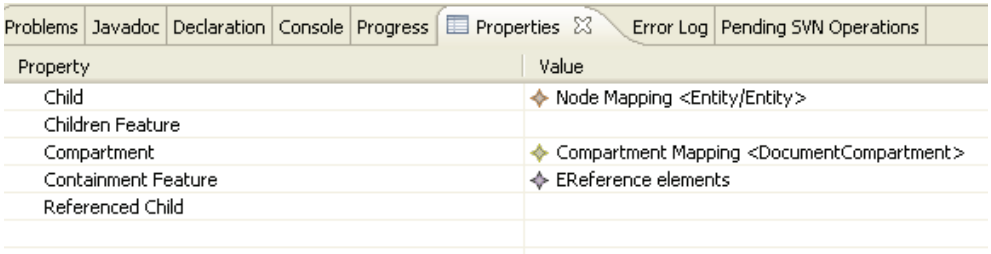

Figure 6.10: Property view of a document's child reference

Link mapping

A link mapping is used to map the connections between diagram elements. Figure 6.11 shows the property view of the link mapping for the element *Association* in the information aspect of MED4SOA.

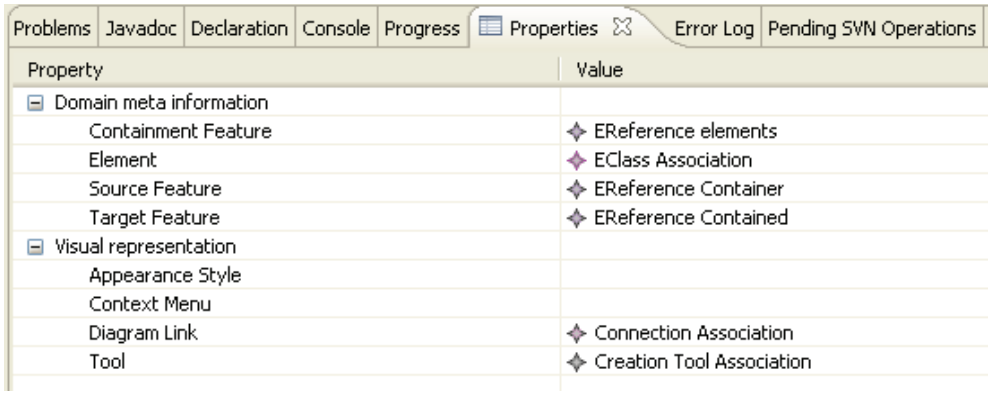

Figure 6.11: Property view of the association's link mapping

Since connections do not have a top or children reference, the containment feature is specified in the link mapping. An *Association* will have a containment feature to be a *EReference elements* and is mapping the diagram link *Connection Association* to the domain element *EClass Association*. In the property view of the link mapping we defined the creation tool and the source and target of the link. The source and target are sett to be the *EReference container* and *EReference contained*, which are both references to *Entities* in the metamodel. In this way we specified that an *Association* will be a link between two *Entity* nodes.

6.2.4 Information generation model

The generation model is the last one created before the generation of the diagram code. This model is created automatically from the mapping model. Its purpose is to regulate the code generation. Mostly, there is no need to alter the generation model

beyond the changing of the domain or plug-in extensions. Anyhow in some cases, additional altering is required in order to complete the development of the editor. Figure 6.12 gives an overview of the elements included in the generation model of MED4SOA's Information aspect.

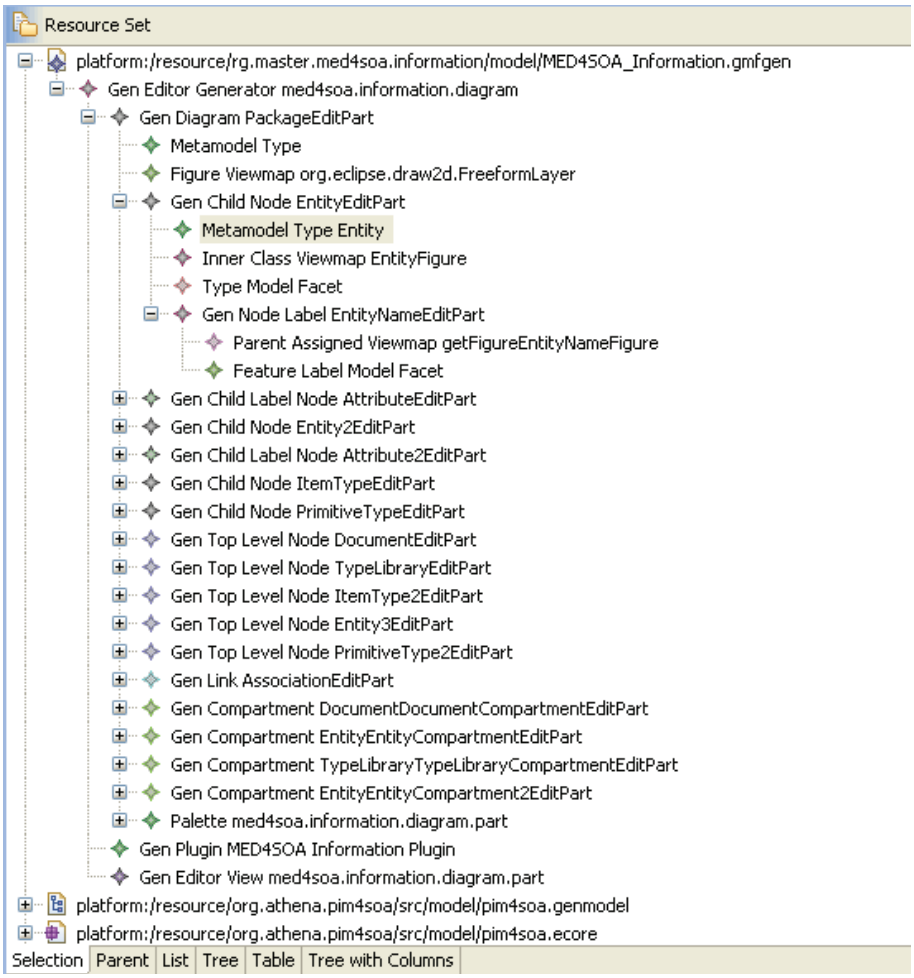

Figure 6.12: The generation model of the information aspect

In the generation model we can identify three main parts. Starting from the bottom, we have the *Gen Editor View*, the *Gen Plug-in* and the *Gen Diagram*. The Gen Editor View has the specification about the future editor, including its ID, package name and icon paths. The Gen Plug-in includes the same kind of information, but concentrates in the generation of the plug-in. The generation specification for the elements of the diagram are given in the Gen diagram. In here, is included information about the generation of palette, top level nodes, child nodes, links, compartments and other parts of the diagram. This specifications are created based on the definition models, the generator model of the domain and other underlying GMF models like the notation or ecore genmodels.

Given the low complexity level of the information aspect of PIM4SOA metamodel, not many changes were made to the generator mode. Since we were creating a diagram for each aspect of the metamodel, the generation target, plug in names and IDs, editor classes, file extensions, model IDs and other similar information needed to be specified in order for the editor to take a complete form. In Figure 6.13 we show the property view of the *Gen Plug-in* of MED4SOA information aspect.

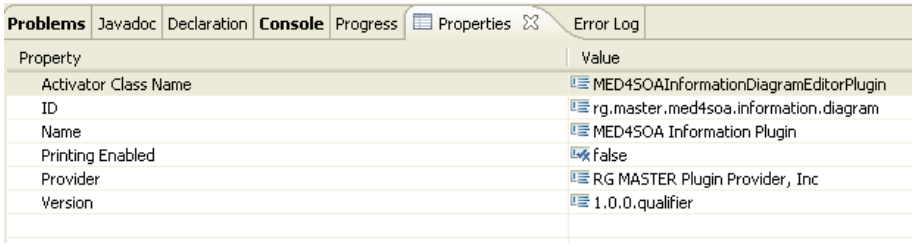

Figure 6.13: Specification of plug-in information in the generator model

6.3 Service aspect

The service aspect of MED4SOA was implemented using the same mechanisms introduced in the information section. We constructed the graphical, tooling and mapping definition model, while the generator model was created from the definition models. As explained in the beginning of the chapter, we will not explain all the details of the implementation of the service aspect nor the process aspect. Rather we will point out some other implementation details not introduces in the information aspect. That being said, we look away from the graphical and tooling definition and continue this chapter with the mapping definition model of the service aspect.

6.3.1 Service mapping definition model

The mapping definition model in the service aspect is more complex compared to the mapping definition of the information aspect. Most of the nodes are composed in each other and the links created in this model realize another kind of abstraction from the metamodel.

Link Mapping

While in the information aspect the only link provided, an *association*, was a realization of the *Eclass Association* of PIM4SOA, in the service aspect all links are realization of *EReferences*. These links will realize some of the connections between classes in the metamodel, being in form of an aggregation or association. Figure 6.14 shows the property view of such a link mapping.

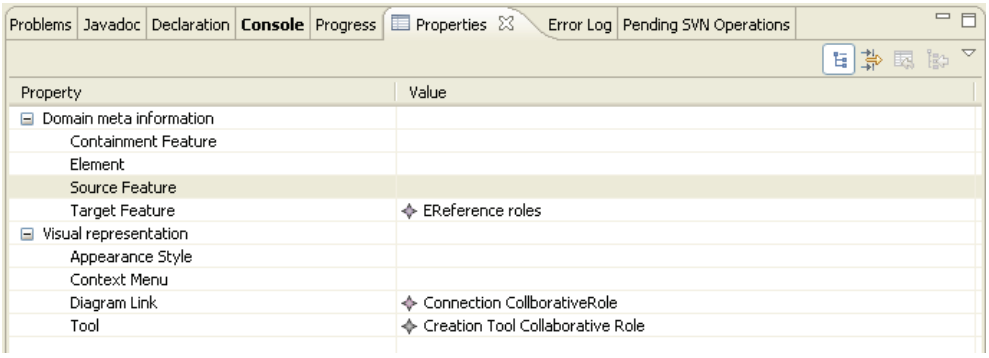

Figure 6.14: Property view of the link mapping for role's aggregation

In the property view we can see that the EReference *roles*, which in the metamodel defines the aggregation of the EClass *Role* into a *Collaboration*, is the target feature of the link. Since the link is defined in the metamodel to refer to a "child"[4] of a collaboration, the containment feature and the source feature are not needed. The visual representations in the mapping are sett to be the creation tool for a *Collaborative Role* and its figure that will be the connection *Collaborative Role*

6.4 Process aspect

In the process aspect we implemented the elements that define the processes and behaviours in MED4SOA. These are realization of the concepts defined in the PIM4SOA process aspect. PIM4SOA does not specify any constraint in the process aspect, or in the other, but in the presentation of the PIM4SOA metamodel in [56], the authors included a note on the *Decision* and *Merge* node. The two notes specify that a decision has only one input and a merge will have only one output. Even though not required, these constraints are realized in the future editor through the *Audition Container* included in the mapping definition model.

6.4.1 Process mapping definition model

Audit container are GMF mechanisms that contain rules defined on diagram elements. In our case we created two audit rules, for decision and merge. These rules are stated in the OCL language, but other implementation possibilities are provided. Since the rules are fairly similar we will only explain the implementation of the decision audit rule.

The audit rule targets a specific domain element, but it is not necessary the targeted element which will be constrained by the rule. In the decision rule we targeted the

[4]containment feature of the EReference that realizes an Aggregation in Ecore

process domain element, in order to state the rule. Without making it more complicated, the rule states that for all flows in a process if there are two flows targeting the same element, and this element is a decision, we imply that these two flows are equal. The rule in OCL is stated below.

$$self.flows->forAll(f1, f2 \mid f1.sink = f2.sink \text{ and } f1.sink.oclIsKindOf(Decision) \\ implies\ f1 = f2\ )$$

In the audit rule the OCL code is specified in a constrained that together with the domain element target form its "children". The audit rule is executed by validation in the running editor and a message is shown in case of a rule not followed.

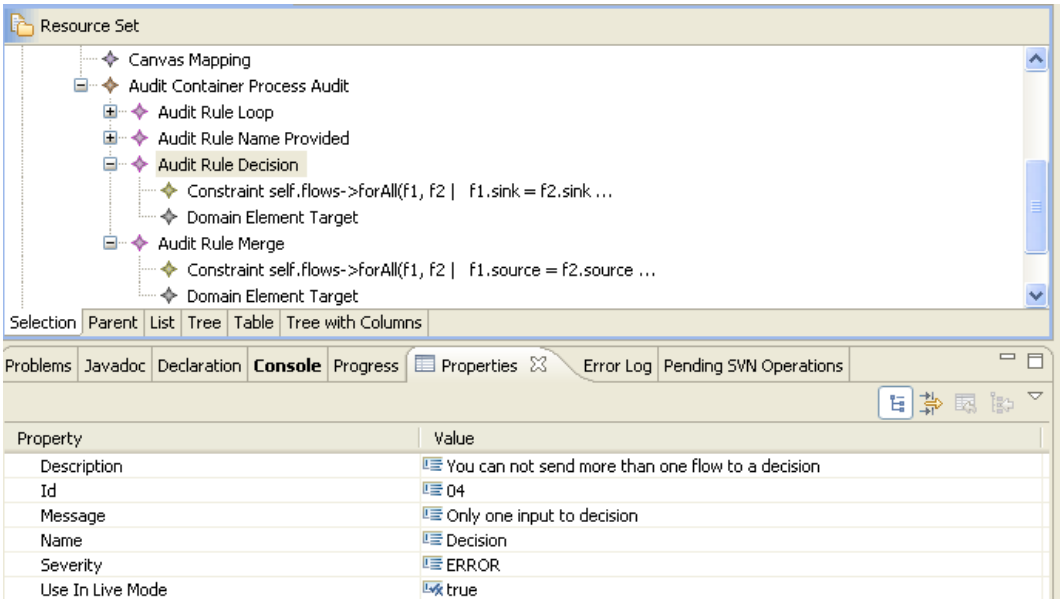

Figure 6.15: Property view of the audit rule for a decision

The validation could be done manually by the editor user or automatically by the editor runtime. The last case is used in the decision rule. It is specified in the *Use in Live Mode* section of the rule's property view to be true. An overview of the two rules and the property view of the *Decision's* rule from the process mapping definition model is shown in Figure 6.15.

6.5 Summary

The realization and implementation chapter explained the work done in the construction of MED4SOA. Concerning the implementation of the metamodel, it introduced the GMF ecore diagram in which we could apply EMF mechanisms to a graphical

diagram in order to create PIM4SOA. Further in the chapter it was explained the realization of the definition models for the Information aspect and how these models connect the domain elements to the diagram or palette elements. Also a short introduction to the generation of code from the GMF generator model was provided. In the end we concluded by explaining the ability of GMF to constrain diagram elements through rules and how these are applied in the future editor.

Chapter 7

MED4SOA example of use

"'And yet,' demanded Councilor Barlow, 'what's he done? Has he ever done a day's work in his life? What great cause is he identified with?' 'He's identified,' said the first speaker, 'with the great cause of cheering us all up.'"

Arnold Bennett

In the following chapter we illustrate an example of use in MED4SOA. We described in chapter 3 the case study, involving the AIDIMA SMEs and their need of a SOA-based system which would increase the interoperability between these enterprises. In our example we will show how MED4SOA is able to give a tool support in the design process and create the models that describe a service oriented AIDIMA system.

If we failed to clarified it so far, the actual model we are creating is a PIM4SOA model, named **aidima.pim4soa**. This is the platform independent model that will describe the AIDIMA SOA-based system. It is important to note that aidima.pim4soa model contains only the semantic data of the model in a XMI standard. It does not include any diagram notation data. The diagram notations are contained in the diagram files. These diagrams are the contribution of MED4SOA, which provides the means to visualise graphically the semantic data.

Another important thing to note in the example is the difference between the different aspects of PIM4SOA and the views of MED4SOA. The three aspects in PIM4SOA defines the elements that describes the information, service and process of a Platform Independent Model (PIM). While the views of MED4SOA are windows to the PIM, they are the diagram editors in which the model is built. The diagrams are initialized from an existing **.pim4soa* model or in the case of a non-existing model, at their initialization a new **.pim4soa* model is created. The diagram notations are included in these views.

We start the description of the ADIMA example with its information structure. Since MED4SOA is meant to be a prototype, we are not modeling the whole AIDIMA system in this graphical editor. Instead we are giving examples of the models created in it and a description of their features.

7.1 AIDIMA information structures

The information structures of a SOA-based system are modeled in MED4SOA using the information diagram editor. The information editor creates a view to the actual model and provides the user with the needed modeling abilities to create diagram elements representing these information structures. Figure 7.1 shows a screenshot of the diagram editor for the information aspect of MED4SOA.

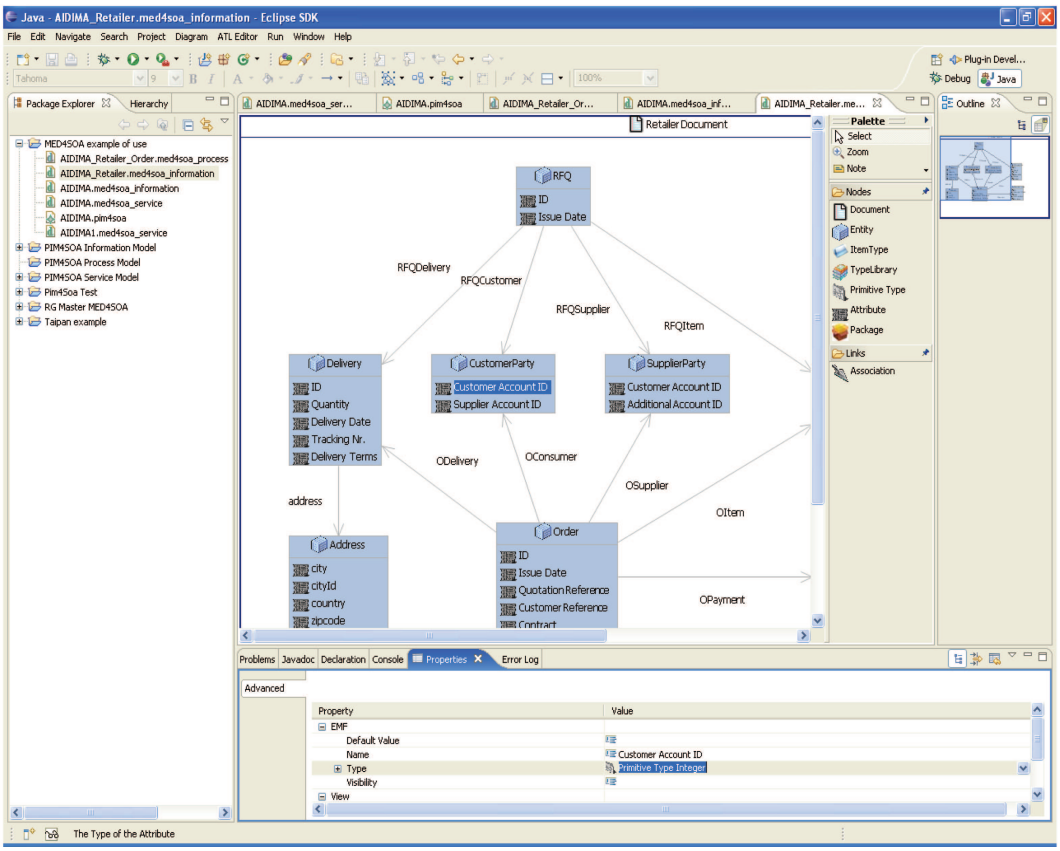

Figure 7.1: The information aspect diagram editor

The information diagram editor has a standard Eclipse layout for the default perspective [1]. The two other diagram editors in MED4SOA, service and process editors, have the same layout as the information editor. These editors are composed of different views. On the left side is the **explorer**, in which projects are listed. Notice in the explorer the difference between the PIM4SOA model and the diagrams opened on it. The diagrams are identified by their file extension, e.g. the information diagram is named *AIDIMA.med4soa_information*.

In the center is located the diagram view, with the palette on the right side. This is in this main view, where the model is created. The creation tools we specified in

[1] Eclipse perspectives and their associated views at [44]

the tooling definition model are now located in the palette. Clicking on one of the creation tools and then on the point of the diagram, will result in the instantiation of a new diagram element.

Other views of the diagram editor are the property view located below the diagram view and the outline. The property view is used to specify features of the diagram elements, which are not shown in the diagram. It is an alternative to the graphical view. The outline is used mostly in modeling wide models. It shows in which part of the model the developer is concentrated. In Figure 7.1 we can see in the outline that the developer is seeing the upper-left part of the diagram.

7.1.1 Documents

The *document* is the centerpiece of the information model. Similar to the other elements of MED4SOA, it has the ability to contain other diagram element in its compartment. In contrary to the other element, the document is the only one which can be decomposed. Since a document is a generalization of a package [2], and the root diagram element on this editor is a package, we can open a diagram editor on the main package or each one of the documents in it. Figures 7.2 and 7.3 shows this decomposition.

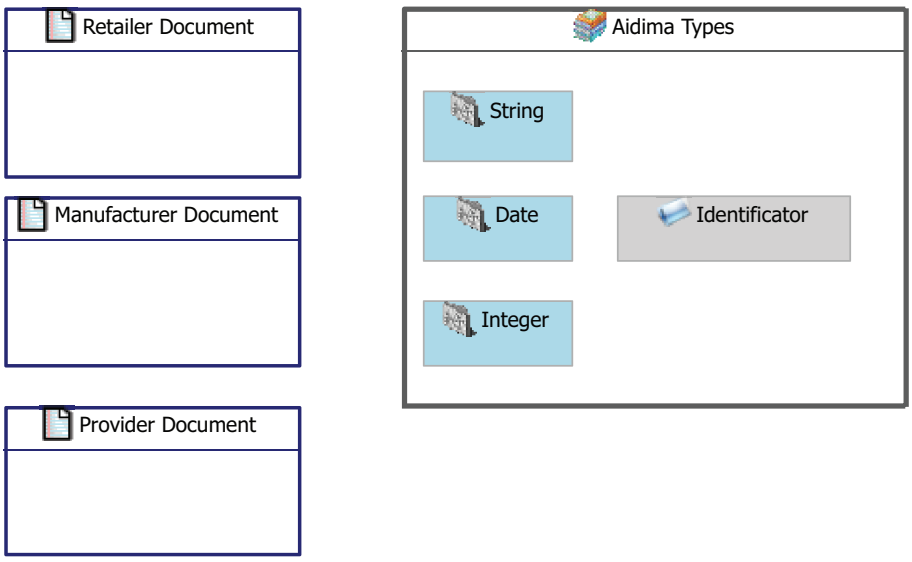

Figure 7.2: AIDIMA information model

In Figure 7.2 we show the AIDIMA information model with its three main documents, one for each of the actors in the system. Beside the documents is shown the a TypeLibrary which contains three primitive types and an item type. The model shows an overview of the information structures of the system. There is first in Figure 7.3 where we can see the decomposition of the Retailer document and have a look at the elements inside it.

[2] referring to the metamodel

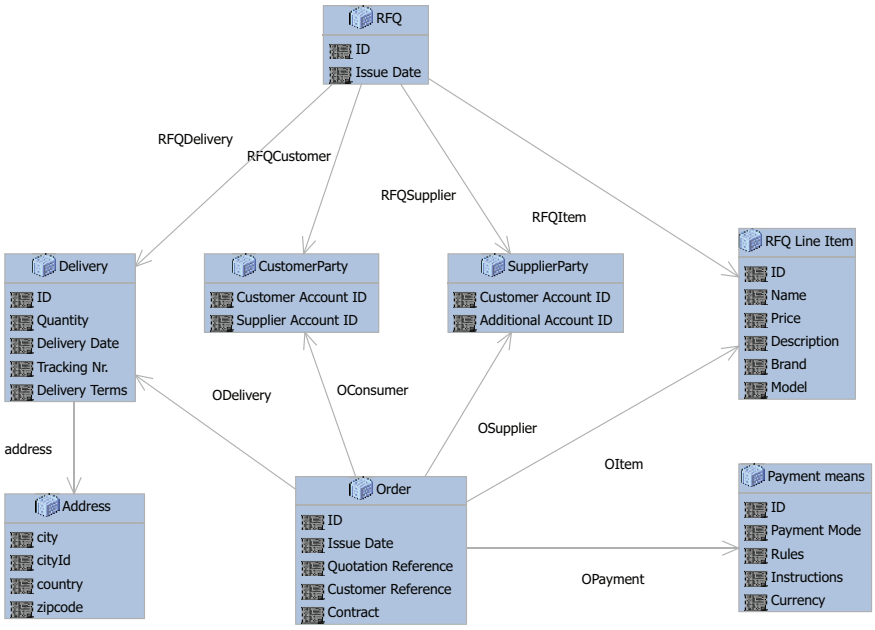

Figure 7.3: The retailer's document in the AIDIMA information model

7.1.2 Entities

The document contains other information structures represented by entities. Entities will represent the type of information that is exchange between a service provider and a consumer. Figure 7.3 shows all the entities contained in the manufacturer document. Note that the Request for Quotation, Quotation, Order and Order Confirmation entities are a simplified version of elements proposed by the OASIS Unified Business Language (UBL) [62].

Entities have attributes. These attributes are defined to have types, which are primitive types included in the type library. The representation of the elements does not show the type of the attribute, but this is defined in the property view for the attribute. In the property view is specified also the visibility of the attribute. Another element in the diagram is the association between the entities.

7.2 AIDIMA services

The diagram elements that define the service and its supporting concepts are modeled in the service diagram editor. In Figure 7.4 we show the diagram editor for the Service aspect of MED4SOA.

Similarly to the information aspect, the service editor has a standard Eclipse modeling layout, with the diagram editor in the middle and the palette on its right side. From the figure we can notice that the creation tools in the palette, contains the names of

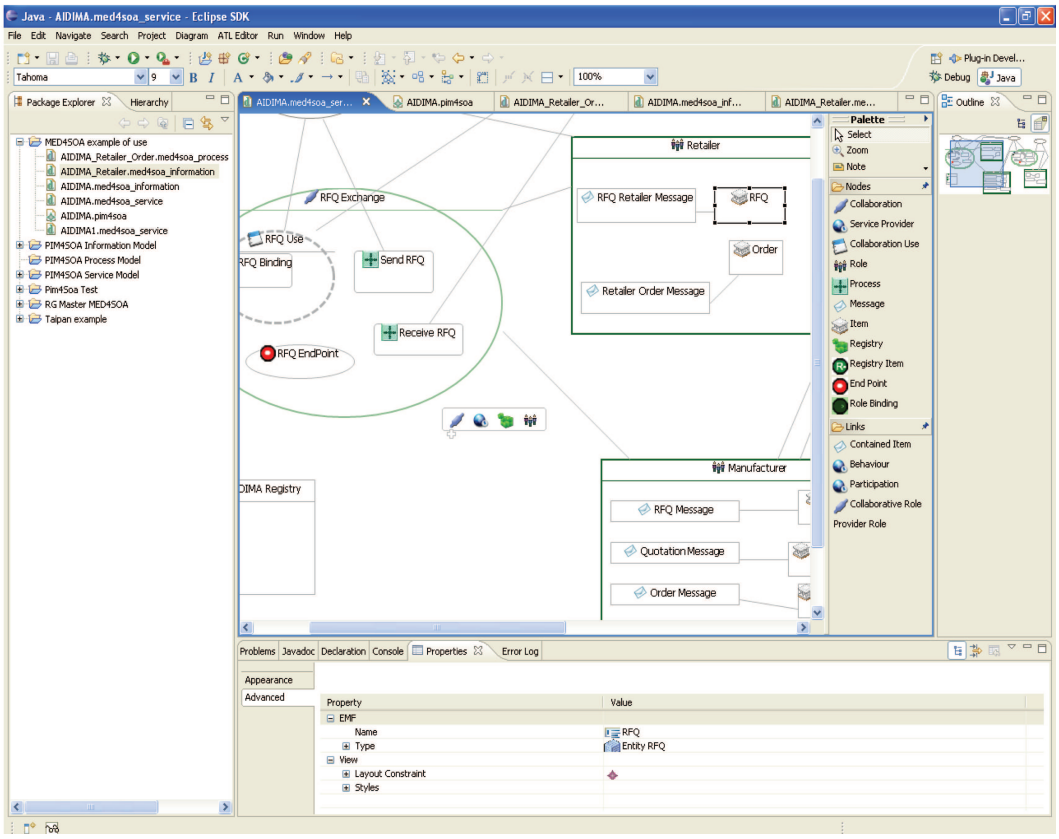

Figure 7.4: The service aspect diagram editor

definitions from the service aspect of PIM4SOA. Another similarity to the information model is the root diagram element. Both information and service model diagram elements are contained in a package. In difference to the information model, the service model contains elements of type PackageableElements, which does not allow us to make decomposing on any diagram element. We describe these diagram elements and the way they form the service model in the next subsections.

7.2.1 Collaborations

When we introduced and explained the concepts of the service aspect of PIM4SOA, we pointed out that the service concept is not explicitly defined in the metamodel. Instead, the service is defined as a collaboration of roles involved in the service. In Figure 7.5 we show a model of two of the services included in the AIDIMA system. These collaborations are named RFQ Exchange and Order Exchange.

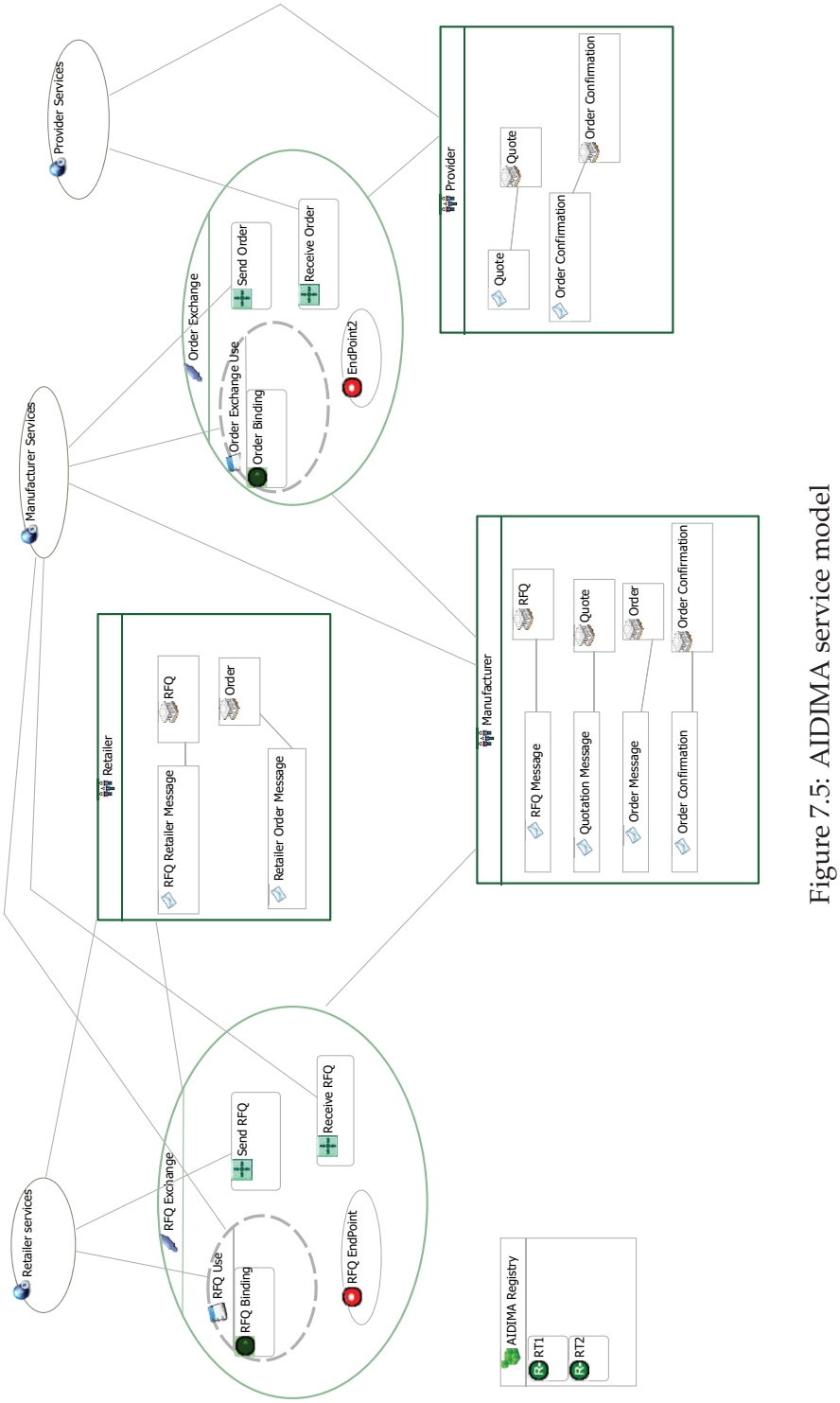

Figure 7.5: AIDIMA service model

The collaboration element is represented in the model as an ellipse with borderlines in a dark see green color. Concentrating on the collaboration *RFQ* exchange, we can see the connection of this collaboration to the roles participating in it. The connection between the collaboration and the roles involved, is defined in the metamodel as an aggregation, while in the model we represent it as a polyline without any labels. Defined as an aggregation, with containment feature sett to be true, this relationship could also been represented as a role contained in the collaboration compartment. To avoid further complexity in the model we decided to show the relationship between the collaboration and its participating roles as a polyline.

The information flow in the AIDIMA scenario, section 3.2, started with an RFQ sent from a *Retailer* to a *Manufacturer*. The retailer and manufacturer are reflected in the model as roles. The role element is represented as a rectangle node with dark green borders. The roles in the system provide the messages that are exchanged. We represent the message as a rectangle node, which is contained in the role compartment. We can notice in the model that a role contains also item elements in its compartment.

The item element constitutes the connection between the service and information aspect of MED4SOA. This element has an attribute *type*, which points to an *entity* from the information model. For complexity reasons the type attribute is not shown in the model. Instead it appears in the property view of the element. In the model we can notice that the *RFQ Retailer Message* is associated to an item named *RFQ*. The *RFQ* type is the RFQ entity from the information model. The link between the message and the item is created as a *contained item*.

Returning to the *RFQ exchange* collaboration, we can notice that it contains different other kind of elements in its compartment. These elements are *RFQ Use*, *RFQ EndPoint*, *Send RFQ* and *Receive RFQ*. The RFQ Use represents a collaboration use. Being a PIM the service model does not specify how the collaboration use is implemented. It only includes a role binding, named *RFQ Binding*. This role binding specifies that the *Retailer* and *Manufacturer* are bound in the *RFQ Exchange*, providing service contract concepts in the model. The attribute role and bound role of *RFQ Binding* are listed in the property view of this element.

The *RFQ Endpoint* represents an address for the service and has its connections to *RT1*, being a registry item. We represent this items as rectangular nodes contained in a registry element, *AIDIMA Registry*. Since a registry is a rectangular node and the same figure visualises also the role element, we chose to use different colors and icons to distinguish these elements from each other. The same approach is used also for collaborations, collaboration uses and service providers, which are all represented as ellipses.

7.2.2 Service providers

A service provider is a specification that describes services(collaboration), roles and constraints(processes). In the model it is represented as an ellipse connected to the three elements named above. For instance, the *Manufacturer Services* describes its participation in a service by having a relationship to the collaboration use *Order Exchange*

Use. It defines the role it is involved in, through a the link collaborative roles[3]. This service provider describes also the processes in services. In the metamodel this elements are represented by a behaviour, being a generalization of a process. In the model we can see that the provider Manufacturer services, is related to two processes in two different services; *Receive RFQ* and *Send Order*. The next section describes the process model.

7.3 AIDIMA processes

AIDIMA processes are modeled in the process diagram editor of MED4SOA. A screenshot of the diagram is shown in Figure 7.6.

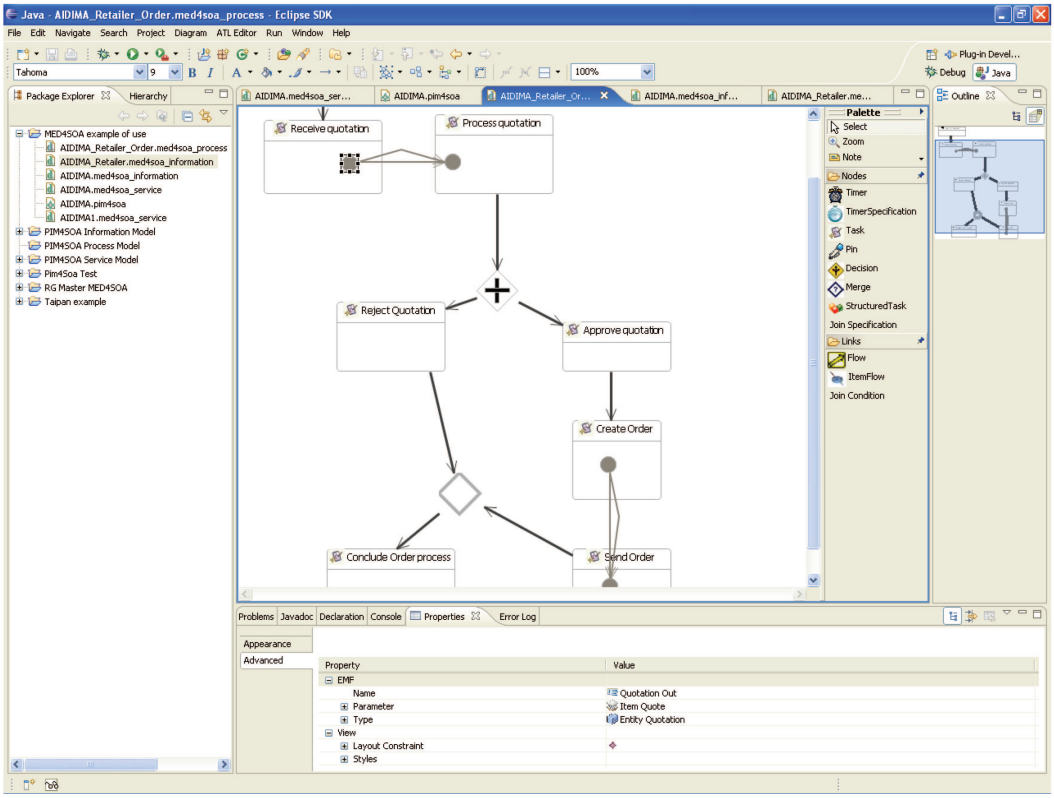

Figure 7.6: The process aspect diagram editor

Similar to the two previous digram editors, the process editor inherit the same look and layout from Eclipse. In difference to them, the process editor does not have a package as its root diagram elements. Instead we specified that all the elements of the process model will be contained in a process element. Since a process is created in a collaboration, we constrain the process model to be build after the service model.

[3]Notice that a link from both service provider and collaboration to a role is created with the Collaborative Role tool in the palette

7.3.1 Tasks

In the creation of the structure for the diagram elements we were inspired from the Business Process Modeling Notation (BPMN) notations [63]. In Figure 7.7 is shown a model of a process in AIDIMA. The process describes the actions taken and the flow of information that happens while making an order.

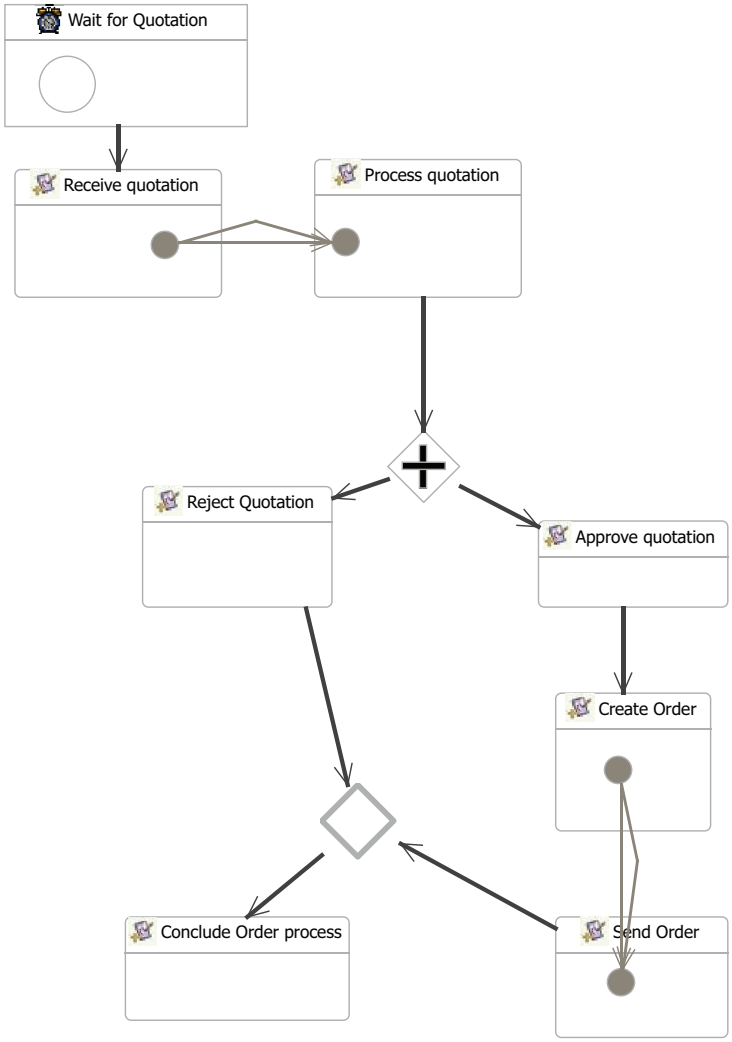

Figure 7.7: The retailer's ordering process in AIDIMA

A timer, which we named for *Wait for Quotation*, waits for a quotation from one or more manufacturer in the system. The timer has a timer specification that will define the end of the waiting and call the next action. The action in the process model are represented by tasks. These are the central elements of the process model. In the figure, the task *Receive quotation* is called from the timer. Both this diagram elements are represented by rectangle nodes, but the task differs from the timer through the rounded

corner of the rectangle, border color or icon. These two elements are connected to each other through flows.

7.3.2 Flows

Flows are represented as a polyline ending in a arrow head. In the model we can notice two kind of flows. The black arrow represents a simple flow, while the double grey arrows represent an item flow. In the case of an item flow, pins are used in a task. In our example when a quotation is received, the next task is to process this quotation, and information is needed to be sent from *Receive Quotation* to *Process Quotation*. The information that will be sent is defined as an *item*. As mentioned in the previous section, the item element is the connection to the information model. This element is specified in a pin as an in or out parameter.

Two other elements in the model are the decision, represented with a cross inside a diamond and the merge element represented with a simple diamond. The decision is an example of a composite figure, where to polylines are used to create the figure. Both decision and merge elements are constrained in their application. We can notice in Figure 7.7 that a decision has only one input and more than one output. The contrary applies to the merge node.

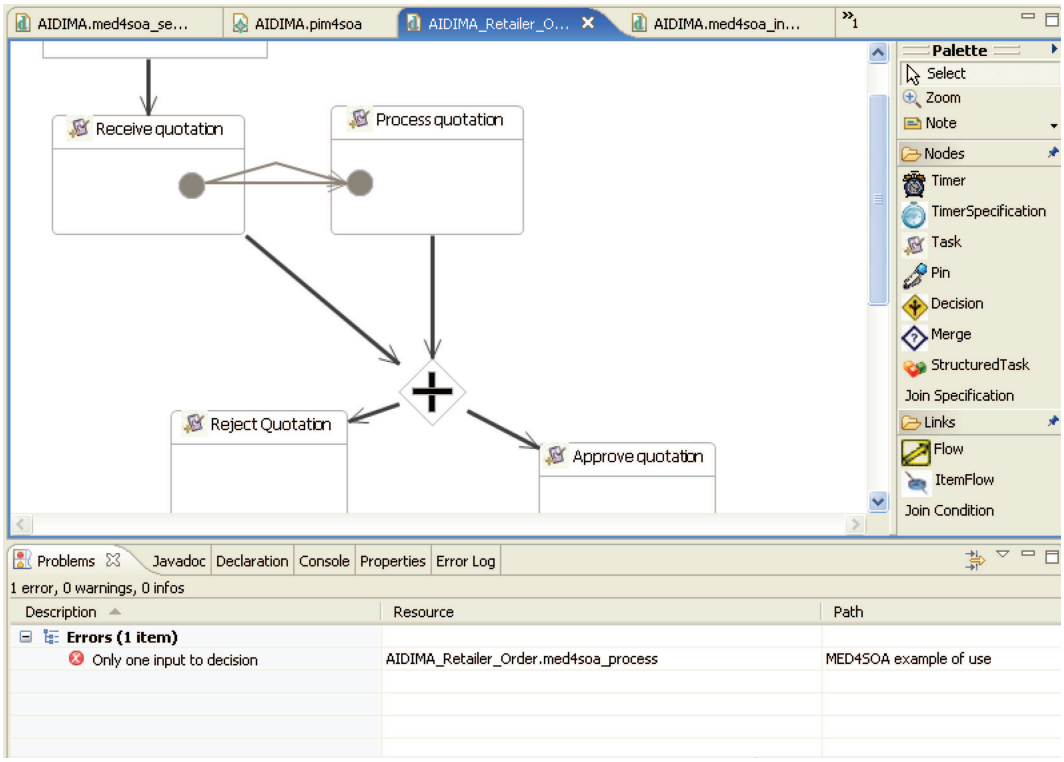

Figure 7.8: Validation of the constraints in the process diagram

In the case of a violation, the OCL constrained we applied to these elements will give an erroneous message upon validation. Figure 7.8 shows the case of a constrain validation. The error message indicates that there are more than one flows targeting the decision. Another option to is to run "live" validation. This GMF mechanism, provided by the Audit Rules, will check the constraints in the runtime and will not create the diagram element, if this element is violating any of the constraints stated.

7.4 Summary

The example of use in this chapter showed the capability of MED4SOA to construct models using graphical diagram editors. Based on the AIDIMA use case, we illustrated the three aspects of MED4SOA starting with the information structures, continuing with the service model and concluded with the process model. In addition, we showed some aspects of constraints and validation included in the process diagram editor.

Chapter 8

MED4SOA evaluation and discussion

"By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest."

Confucius

DURING the development of MED4SOA, the experience gathered is important to us and others whom research is based in metamodeling, creation of visual languages, tool support for these language or specific architectures like SOA. In this chapter we discuss our experience with PIM4SOA and the GMF technology.

But first, and most importantly this chapter will evaluate the hypothesis with respect to the success criteria previously defined in chapter 3. Every success criterion, is followed by a short argumentation on its validation result.

8.1 Fulfillment of success criteria

We validate the success criteria through a level of fulfillment. These levels are Fulfilled, Partly Fulfilled and Poorly Fulfilled. In addition, we use a graphical notation for these levels: ✔ for Fulfilled, ❍ for Partly Fulfilled and ✗ for Poorly Fulfilled.

8.1.1 Success criteria

The Metamodel-based Editor for Service Oriented Architecture (MED4SOA) shall:

Success Criterion 1

P1: *provide means to construct models **graphically**.*

✔ **Fulfilled**

The construction of MED4SOA and the example of use provided in chapter 8, showed that it is fully possible to create a model graphically. Initialization mechanisms included in MED4SOA gives the opportunity to start an editor on the **.pim4soa* file. With the help of the palette on the right side as shown in figure 7.1, 7.4 or 7.6 we could create the diagram elements. Together with other features that MED4SOA inherited from GMF, the user has the freedom to move, edit, save or delete these diagram elements.

Success Criterion 2

P2: *realize graphically **all** the concepts and abstractions forming the semantic data in a SOA metamodel.*

✔ **Fulfilled**

Each and every domain element of PIM4SOA metamodel was realized graphically in MED4SOA. Since the implementation of the metamodel was achieved through EMF technology, and recalling that GMF is an extension of EMF features, we can conclude that GMF is created with the purpose to realize graphically these elements. Being also an extension of GEF, the graphical editor provides even more elements which are not defined in the metamodel, but are necessary in creating the concrete syntax of a modeling language.

Success Criterion 3

P3: *provide means that will make possible the representation of the semantic data in different contexts. These contexts could be different views or graphical representation of the concrete syntax.*

✔ **Fulfilled**

The semantic data are indeed represented in different contexts. EMF utilizes *EClasses, EReferences or EAttributes* to implement the metamodel. For instance the *EClass* element is realized in all three editors and with different figures. The example in figure 7.4, shows a metamodel class element represented as an ellipse node in a *Collaboration*, or a rectangle node in a *Role*.

Another way to represent the elements is the property view or the tree-editor in which EMF relied before the introduction of GMF. The tree-editor is associated with the metamodel and could be used to create "models". From our point of view, even if it does not make any sense to practice the tree-editor instead of the graphical editor, it shows once again the diversity of MED4SOA in the representation of semantic data.

Success Criterion 4

P4: *be applied to create persistent models that have the ability to display semantic data multiple times in the same diagram.*

✔ **Fulfilled**

In the architecture and design chapter, we mentioned that it is EMF, which furnishes the graphical editor with the persistency mechanisms. The semantic data of the models are stored in an XMI file, while diagram notations are stored in separate diagram files.

Semantic data are displayed multiple times. It is GMF that provides this functionality by assigning a notation model on the runtime to each open diagram. The notation model stores the information about diagram elements and assures their plurality in the diagram. Figure 7.6 is an example of the multiplicity, where we can notice the *Task* node or the *Flow* link multiple times in the diagram.

Success Criterion 5

P5: *promote simplicity in the design process. The simplicity covers both the aspect of comprehension and structure of the models created.*

❍ **Partly Fulfilled**

With only three types of diagrams, MED4SOA can be defined as simple tool. In the same time it provides elements from definitions that are familiar to software architects of business analysts. It has a relative small number of diagram elements and the creation or management of these elements resemble other major industrial modeling products, e.g. RSM.

The reason for not fully accomplishing this criterion is the complexity of the service model. Due to the semantics of elements in the service aspect of PIM4SOA metamodel and the close coupling between these elements, we could not provide a better solution to this model. The model could not be decomposed as the information model and in the case of a larger enterprise, we believe that the complexity of this model could be an issue.

Success Criterion 6

P6: *represent the domain elements in conformance to their semantics and notations applied in the modeling world.*

✔ **Fulfilled**

We created the graphical representation of the domain elements inspired by other notations, which are part of modeling standards like UML or business standards in the case of BPMN. Examples of the notations inspired by UML are figures 7.1 and 7.4, where respectively the *entity* and *collaboration* resembles the *class* and *collaboration* elements of UML2.0.

The semantics of the metamodel elements are also preserved in the visual representation. For instance the *association* element in the information aspect or *flow* in the process aspect, does include features [1] that imply connection to other elements in the metamodel. Based on this features, the diagram elements of *association* or *flow* are links, visualized as polylines. Hence, MED4SOA fully accomplishes this success criterion.

Success Criterion 7

P7: *provide the user with clear and distinguished diagram elements in the model.*

✔ **Fulfilled**

The diagram elements in MED4SOA are representation of figures inherited from the figure gallery. The figure gallery is included in the GMF's graphical definition model and provides a variety of figures from ellipses to rectangles or polylines. Even though not in great number these figures allow the user to distinguish most of the diagram elements from each other. For the rest of the elements that are represented by the same kind of figure, the differentiation is achieved by the variation of the element's color or its icon. In this way we provide clear and distinguished diagram elements in the model.

8.1.2 Hypothesis

H1 *A customised graphical editor for a high level DSL will be an efficient tool to design SOA-based systems.*

✔ **Validated**

Through the evaluation of the success criteria, we have validated the hypothesis **H1**. From the seven success criteria that acted as predicates to test the hypothesis, six were **Fulfilled**, while only one was **Partly Fulfilled**. This result, led us the conclusion that **H1** is true. From here we can confirm that the main goal of this thesis is accomplished. Now we can state that :

> *"Metamodel-based Editor for Service Oriented Architecture (MED4SOA) is an efficient tool to design SOA-based systems."*

Thus the hypothesis is true, there is a main thread to its validity. In the last chapter, we showed examples that proved the ability of MED4SOA design SOA-based systems. From the success criteria we listed, we proved that MED4SOA is efficient in designing these systems, but the main thread relies in the understanding of the word *efficiency*.

We argued in section 3.6 on our understanding and use of the word *efficiency*, but the lack of a common understanding in its meaning can make a thread to the validity of *H1*. Defining efficiency in another aspect, e.g. time or energy spent aspect, or even test

[1] contained, container in associations or source, sink in flows

the hypothesis using other success criteria, may or may not lead to the same results. This thread can not go unseen and we recommend further testing and evaluation of MED4SOA.

8.2 Discussion on the metamodel

In this section we will discuss our experience on using the metamodel in relation to MED4SOA. We will have a look back at the requirements for the SOA metamodel, and give our recommendations in its further improvement

8.2.1 Review of the metamodel requirement

In the beginning of the development of MED4SOA, we *believed* that PIM4SOA is a satisfying metamodel to lay the fundamentals of a DSL for SOA. The score it received in section 4.3.1 lay the roots of this belief and its construction by using MOF-based technologies makes this metamodel easy applicable to graphical frameworks. Most of the candidate graphical frameworks, which would realize the visualization of the elements of PIM4SOA accept their metamodel input as MOF constructs. This gives PIM4SOA the advantage to be the abstract syntax of a modeling language that could be visualised graphically in more than one framework.

Further, being a platform independent model that promotes **business-orientation**, loose coupling and with the needed expressive power, it is able to define the elements that could constitute a model of a SOA-based system. Even though this characteristics did not receive the highest score, we provided examples in the previous chapter that confirm its ability to model SOA-based systems. The business-orientation can be noticed in the information and process aspects of MED4SOA, where most of elements are inspired by EDOC and BPDM. These elements like *Document, Entity, Task* or *Flow* provide the fundamentals for modeling the information structures and processes inside an enterprise or between different enterprises.

Loose-coupling of services, is achieved through elements like *Collaboration* and *CollaborationUse*. The service is not explicitly defined in the service aspect of PIM4SOA and its definition by the two elements above may inflict unnecessary complexity to the model. The expressive power of the service aspects is rich enough to describe the main concepts of the service domain, but as we based ourselves in the OASIS Reference model for SOA, the service metamodel lack the ability to define other detailed service elements. We found abstractions like *service visibility*, *service functionality* or some aspects of *service contract & policy*, difficult to model with the existence elements of the metamodel. Furthermore, the structures of the metamodel are not well described. In order to have a clear definition of this structures, more semantic definitions are needed.

The simplicity it provides, in comparison to the other alternative metamodels, gave us the opportunity to have a better comprehension of the metamodel. This resulted in closer familiarity with the metamodel, that made it possible to keep the focus on the

righteous visualization of the elements and not in the semantic details of the metamodel. Stated in another way, assuming that we would have chosen another metamodel for MED4SOA, far more complex and with far more elements, we could have lost the focus on the actual graphical editor we were implementing, and concentrate ourselves in the understanding of the semantic details and definitions of the metamodel.

In conclusion to this part of the discussion, the extensibility of PIM4SOA may be an issue in the future to come. We have experienced during the implementation of MED4SOA that indeed it has the capacity to evolve and be extended in the future. But how far would this extension go and how much reconstruction of the metamodel would be needed? In PIM4SOA the basic elements which the rest inherits from, are only four [2], and in the future these elements may or may not be enough to provide the basics for the extension part. Additional elements would be needed to back up the extensions, which could be aspects of *events, behavioural rules* or *security*. Another dilemma in this case would be to find out where is the limit for the extension of the metamodel, where it ceases to exist as a metamodel for a specific domain and grows to be of a "general purpose". In addition, the reconstruction needed in the extension process would result in a complete transformation of PIM4SOA beyond recognition. Will we end with an evolved metamodel or a whole new one?

In the beginning of MED4SOA's development, we **believed** that PIM4SOA was the founding stone for the DSL that will be modeled in it. Now that the development is complete, we are indeed convinced that it is the right metamodel to start the development of such a tool. It is not the perfect metamodel, but it is good starting point and its drawbacks have full potential to be covered.

8.2.2 Recommendations

Except the changes made to the PIM4SOA metamodel, we have some more recommendation in the aspect of its expressiveness and definition of services. Our experience in modeling in all the aspects of MED4SOA led us to these recommendations. They aim to improve the functionality of the metamodel and its capability to grow larger than only the fundamental metamodel for SOA.

From our experience we believe that PIM4SOA should:

- include more definitions concerning the dynamics of the service
- include more definitions on the service descriptions, most importantly concerning the service contract
- define constraints in all aspects of its metamodel

[2] Element, NamedElement, Package and PackageableElements

In the OASIS reference model for SOA, it is pointed out that the dynamic interacting of services supports a set of concepts that are about the service itself. The *service execution concept* and part of *service description or service contract & policy* are missing in PIM4SOA.

The service execution concept, is not fully included in PIM4SOA. Indeed *Collaboration* and *CollaborationUse* defines part of the execution, like the role binding, registry items or end points, but main concepts of service interactions are not explicitly defined. In our model of AIDIMA system we failed to recognize abstraction of PIM4SOA, realized in diagram elements, that could model infrastructure elements, like interaction points or interface specifications. This abstraction may be necessary in the description of service execution.

The service aspect, still in regard to OASIS reference model does not entirely describe the service description. We mentioned above the visibility or functionality of the service, and we could add the concept of reachability. PIM4SOA provides concepts of end points and registry items, which identifies addressable services, but definitions of interaction protocols and channels, may help to complete the picture of the service. Service contracts should also be included to help the service provider have a clear definition of the service offered.

Another important element we failed to identify in PIM4SOA are *constraints*. This concept may be applied in different parts of the metamodel. Constraints may be needed in the flow of information, operations or service interaction. Even though constraints may be implemented in the process of defining the concrete syntax of a DSL based in PIM4SOA, we believe that the explicit definition of constrain abstraction in the metamodel may help the specification of service policy.

Our recommendations on PIM4SOA come only from our experience in realizing the concepts of this metamodel in MED4SOA. Parallel to the development of MED4SOA a group of OMG's submitting organizations and other contributors submitted lately a UML Profile and Metamodel for Service- for Heterogeneous Architectures(UPMS-HA) [64]. This initial submission, which is mainly concerned in a standardised metamodel for services, bases its solution on PIM4SOA. Even though our work has undergone independently from this submission, we share the same ideas in the changes applied to PIM4SOA[3].

The timing and the inclusion of PIM4SOA as basics to this initial submission, shows once more that this metamodel lays indeed the fundamentals of a metamodel for services.

8.3 Discussion on the editor

Next, we share our experience in the implementation of MED4SOA. We point out some features of our solution that are worth the discussion, and concentrate on some

[3]Note that even though SINTEF is a supporter in UPMS-HA, our work did not have any impact in the changes applied to PIM4SOA, nor did we contribute in the application of these changes

advantages or drawbacks of GMF technology. We have explained the details of the editor from the architecture chapter, through realization, example of use and in the end on the argumentations given to the fulfillment of the success criteria. Not much is left to say, but anyhow we will try to touch some delicate areas of MED4SOA and its implementation technology.

8.3.1 Changes on the metamodel

In chapter 4 we introduced small changes to PIM4SOA. These changes did not have any effect on its purpose, but made it easier to realize the metamodel in GMF. In here we discuss the changes made in the containment feature of the aggregation in the metamodel. We should also have made some changes on the structure of the metamodel, but for some reasons named below we decided not to.

Containment

The under layer of GMF that takes care of persistency, namely EMF, stores the model we create in a XMI file. Being a flavour of XML, the XMI format saves the stored information in nested structures. This kind of structure becomes an issue when an element is defined to be contained by two different metamodel elements, and result in an erroneous structure of the model leading to a runtime error in GMF.

For instance, if a behaviour[4] in the service aspect will have an aggregation from both collaboration and service provider, when we would define the connections to the behaviour, the EMF runtime mechanisms will store two instances of a behaviour, one under collaboration as a constraint, and the other under service provider as a behaviour. This leads to an unwanted redundancy and runtime error, which we simply resolved by the changes in the metamodel. Things will get more complicated in case of a standardized metamodel, where changes are not allowed and there exist multiple aggregation on one element.

Package

Another issue in the implementation of MED4SOA is the definition of the concept of *subpackage* in a *Package*. The advantage with the subpackage, is the fact that all models can be included under one package. In the case of a future transformation to a Platform Independent Model like Web Services or BDI Agents, this feature of the metamodel will result to be very helpful, since it simplifies the input, reducing it to one. But this cause an issue in the realization of the metamodel in GMF, given that two of the views have a package as the root diagram elements.

Having a package as the root diagram element for two views can be a problem in the fact that we could not constrain the view on which root diagram element it should open. For instance, at the initialization of a service diagram, if we choose a *Document*,

[4]generalization of a Process

which is a package, to be the root diagram element, we end up with a service diagram view on a *Document*. This does not make any sense, because service diagram elements can not be added in a document. Constraints on the diagram level are needed in this case, but given the maturity of GMF, we believe that it is still early for this technology to cover this problem.

A solution, as we practiced above, could be changes in the metamodel. The division of the metamodel in packages, and the important role that the *Package* plays in the metamodel did not allow us to apply any changes. The division of each aspect of the metamodel in packages, gives it an open perspective in the future for the extension with other aspects. The *Package* is a central element of the metamodel and we decided not to change the metamodel since it could result in a radical change.

8.3.2 Constraints

In the discussion on the metamodel, we pointed out that PIM4SOA lacks the ability to define constraints on the model. This could be a drawback of PIM4SOA in regard to business orientation, where rules and constraints are central concepts. Anyway, GMF provides a way to repair this by introducing constraints on the diagram elements. These constraints are not based on the metamodel but mostly in the semantics of the elements which are constrained.

An example is the constraints on the elements *Decision* and *Merge*, explained in section 6.4.1. In the ATHENA IP A6 presentation of PIM4SOA the constraints on decision and merge were specified as a note attached to the metamodel elements. Applying the constraints introduced in GMF, we managed to complete the semantics of the elements in the metamodel. This shows that GMF is not only a generic framework for graphical editors, but aims also to provide additional elements to the future model.

The ability of GMF to sett constraints on the diagram elements in the editor, brings this technology further than the standard node and edge representation of elements. Even though not explicitly supporting business rules, in the very near future GMF has all the potential to achieve this status, and catapult itself to the front of the progressive technologies in graphical editor development.

8.3.3 Development methodology

It may sound odd that we mentioned the development process of MED4SOA in section 5.4, but anyhow did not touch the aspect of the method used to develop this editor. Except the research methodology, we could have defined a method for the future researchers. They could have based they development of their tool in this method. We deliberately decide not to include in this thesis a MED4SOA method.

Figure 5.10 is a well known diagram in the GMF society. It explains the development process of an editor in GMF. With few changes, this development process could be evolved in a specified version of a development methodology, e.g. MED4SOA

Method. Due to our respect for the developers of the Eclipse GMF Project we did not define our own methodology, but instead used the GMF development methodology.

Any how this thesis in its entireness describes in detail how to proceed in constructing a customized editor for a DSL. Even though not explicitly defined, a researcher or developer can find useful the work described in this thesis to construct its own product. To support the development process in the latest version of GMF, [5] is included a dashboard. This dashboard will help the developer work through the flow of producing a GMF-based editor, by invoking actions for the steps that are often used in the process. More is to find in the latest GMF tutorials in [65].

8.3.4 GMF usage

Working with GMF, we experienced that it is an innovative technology with high potential. GMF can result to be a leading technology in graphical modeling, as it can implement editors supporting a DSL in a short time and with low implementation effort. Being a generative framework, GMF provides automatic generation of graphical editors, supported by the MDSD approach. This quick and low effort implementation of tool support could be a big aid in the software industry.

As mentioned in the evaluation of the success criteria, MED4SOA through its implementation technology GMF, has the capability to provide the representation of all the metamodel definition, in simple or complex figures. It can also achieve the decomposition of certain elements in the diagram, but lacks the ability to add behavioural aspects of generated figures. Yet, GMF does not have ability to generate complex GEF figures that listen to events and initiate actions based on these events. In the case of complex composite figures, like the UML sequence diagrams representations introduced in SeDi[66], GMF could support the decomposition of the figures, but this decomposition will not be automatic. The automatic decomposition could be achieved by assigning the behavioural aspect described above, which is not yet implemented in GMF.

The main drawbacks of this technology are its immaturity and the lack of supporting documentation for developers. MED4SOA was implemented in GMF 1.0.1, but in the last months(2007) the 2.0 version was launched. In an attempt to migrate to version 2.0, we experienced frustration, difficulties and did not succeed in our attempt. Even though there are none known issues in this migration [43], the code generated from migrated definition and generator models resulted erroneous. We believe that the issue relies in the assumption of the GMF generator about generation target specifications that begin with a capital letter. These target specifications are processed as Java classes. In our projects, the packages containing implementation of the metamodel, have their named beginning with a capital letter. These packages are assumed to be Java classes by the generator, and imported in the generated files as classes and not packages. Due to time shortage we did not have the opportunity to continue the investigation of this issue.

[5] version 2 M4

Concluding with GMF we recommend the usage of this technology to boost up the creation of editors that supports a given metamodel. Even though capabilities of the editor do not go beyond a certain levels of complexity, i.e graphical models, validation, rules, manual decomposition of diagram elements, this editor provides extensible features. It builds on top of the Eclipse platform and additional functionality to the generated editor can be included through the plug-in architecture.

8.3.5 The way ahead

Before we conclude the discussion, we want to add some words on the future of the models created in MED4SOA. Using the principles of MDA, there should be a CIM that is transformed to a PIM and further on to a PSM. The first transformation is very complex and there is still no concrete transformation that could bring all the abstractions of the business model to a PIM. But the transformation from PIM4SOA to a PSM is a reality. The developers of PIM4SOA have included in the project site a transformation to WSDL [34]. Also a transformation to BDI Agents is presented in [38].

These transformations from PIM4SOA, show that the graphical editor we provide is not a figurative tool in which we can construct platform independent models of a SOA-based system, but can actually be an important tool in the MDA chain of transformation. In the waiting for a CIM to PIM transformation, the developer could use the CIM as conceptual basics to create a PIM in MED4SOA, and continue further with transformations to WSDL or BDI Agents, toward the concrete implementation of the system.

8.4 Summary

In this chapter we evaluated the fulfillment of the success criteria for MED4SOA. The evaluation of the success criteria showed that all except one criterion, were fulfilled. We concluded that the hypothesis **H1** is valid.

Further, we discussed our experience with the metamodel, PIM4SOA and the technology used to realize an editor based in this metamodel, namely GMF technology. In the discussion, we looked back at the requirements for the metamodel and argument why PIM4SOA is the right metamodel for MED4SOA. In the context of the metamodel, based on our experience we recommended some changes to PIM4SOA in order to improve the functionality of the metamodel.

In the discussion on the editor we looked at some difficulties in the implementation of containment features and packages in GMF. We recognized the ability of this technology to sett constraints on diagram elements and pointed out the tool support for the development method of a graphical editor.

Chapter 9

Conclusion

"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning."

Sir Winston Leonard Spenser Churchill

THE development of MED4SOA resulted in the construction of a prototype, which lays the fundamentals of designing SOA-based systems with a DSL. The abstract syntax of this DSL is defined in PIM4SOA, while the concrete syntax is implemented in the prototype we provide.

The main objective of the thesis was to provide the developers with a prototype in which they could design SOA-based systems. We described MED4SOA throughout this thesis and successfully validated the hypothesis H1 :

MED4SOA is an efficient tool to design SOA-based systems.

Under the investigation the hypothesis, we identified the two main parts of MED4SOA: the metamodel and the graphical framework in which the prototype is implemented. The result of the evaluation of different metamodels that capture the service domain, showed PIM4SOA to be the most suitable metamodel in which we could base the editor. PIM4SOA received the highest score, mostly due to its simplicity, but still having the needed expressive power. In the aspect of the graphical framework, the key characteristic that turned out to be decisive in the choice of an implementation technology, was the open source trademark. GMF was chosen to be the implementation framework for MED4SOA.

Through an example of use we showed that MED4SOA provides means for constructing models graphically. The elements defined in the metamodel are represented graphically using GMF and its underlying technology GEF. Based on the different aspects of the metamodel, MED4SOA structures the diagram elements in three different diagrams: information, service and process diagram. These diagrams are provided through their respective editors, which use the GMF runtime to visualize the elements and EMF facilities to provide persistent models.

From the experience gained in the development of MED4SOA we can conclude that PIM4SOA provides the basic concepts of the service domain and should be used as the fundamentals of a DSL for SOA. The GMF technology from our point of view, is still immature since it lacks the stability and documentation [1]. But during the development process, we experienced that it provides all the capabilities for constructing graphical editors. Even more, with the inclusion of constraints and audit rules, it provides additional functionalities to the graphical editor.

9.1 Achievements

The development of a Metamodel-based Editor for Service Oriented Architecture:

- identified the two main pieces of a graphical modeling editor, namely the metamodel that describes the concept of the problem domain, PIM4SOA, and the graphical framework that will implement the visualisation of these concepts in the editor, being GMF.
- developed a customized graphical modeling editor, which provides the needed features to design SOA-based systems.
- revealed PIM4SOA to be the fundamental service-oriented metamodel and pointed out the need for further development of this metamodel, especially in the context of service dynamics, service description and constraints.
- investigated the efficiency and capabilities of GMF as a graphical modeling framework, revealing the potential of this framework to build, generate and run graphical editors.
- provided a precedent in the development of a graphical modeling editor for SOA, which lays the roots for further development in this field

9.2 Future work

From our experience in the development of MED4SOA, we recommend further development on this graphical modeling editor. Since it is developed as a prototype, MED4SOA should be tested in a development project and evaluated for further improvement. Our example of use may have shown the efficiency of constructing models based on PIM4SOA, but we believe that a more practical testing is needed in order to explore the full potential of MED4SOA. Another important aspect of the development, is the usability of this editor, which we did not have the possibility to improve.

We recommend the further continuation of MED4SOA's development, in order to achieve a customized editor for the new SOA metamodel that is proposed lately in the UPMS-HA submission. The development of PIM4SOA should continue in order

[1] we are referring to GMF 1.0.1 in early 2007

to provide additional semantics to the model, i.e. semantic services and semantic interfaces. Also definitions facilitating the modeling of variability, configuration, agents or events should be included in this metamodel. The evolved metamodel may or may not reassemble to the existing PIM4SOA metamodel, but anyhow we believe that the precedent provided in this thesis may be helpful and provide the fundamentals for the future graphical editor.

This editor should also undergo a comparison to another UML modeling tool. IBM's RSM is a UML tool that provide features to design SOA-based systems using UML profiles. We recommend an evaluation of MED4SOA compared to this tool. The evaluation should test the comparison to the efficiency of DSL vs. UML profile, but in the same time compare the UML metamodel against the future metamodel for SOA. Comparison to editors constructed using Microsoft's Software Factories may also be needed in the near future.

We also recommend the definition of a design methodology for MED4SOA. This methodology should guide the developer in the construction of models in this tool, and even further in future implementation of transformation applications.

Appendix A

PIM4SOA tree-like metamodel

Appendix A shows PIM4SOA in an Ecore tree-like model. The following screenshots aim to provide the details of the metamodel that are not visualised in the graphical models of section 5.2.

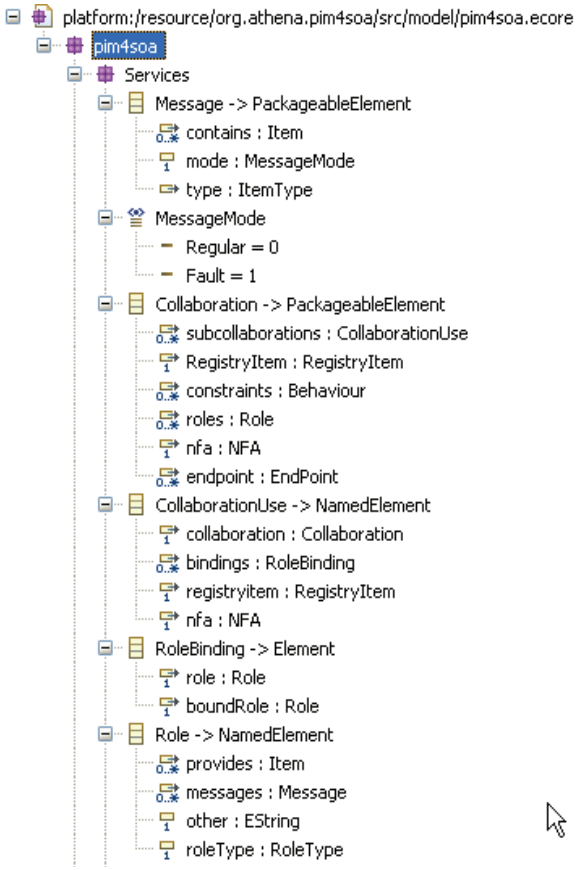

Figure A.1: The service aspect of PIM4SOA in an ecore tree editor

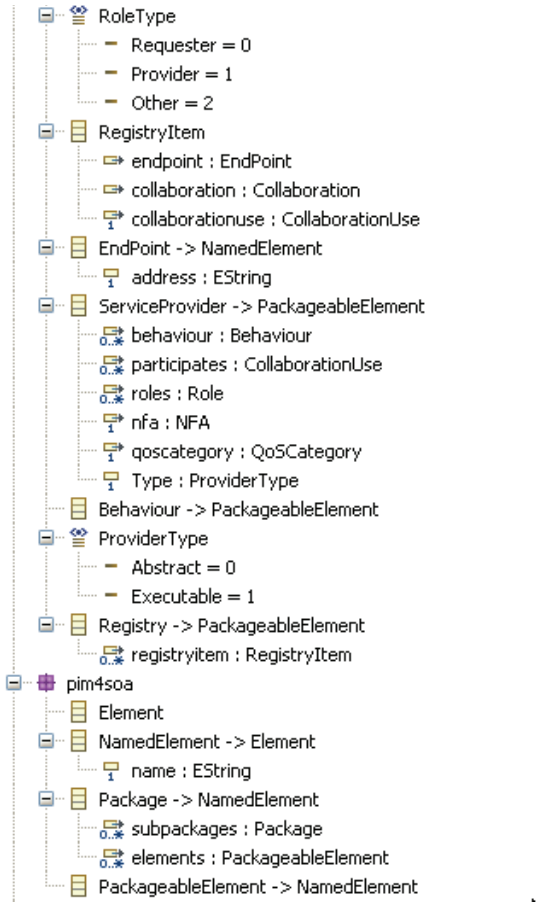

Figure A.2: The continuation of the service aspect of PIM4SOA with the general elements, in an ecore tree editor

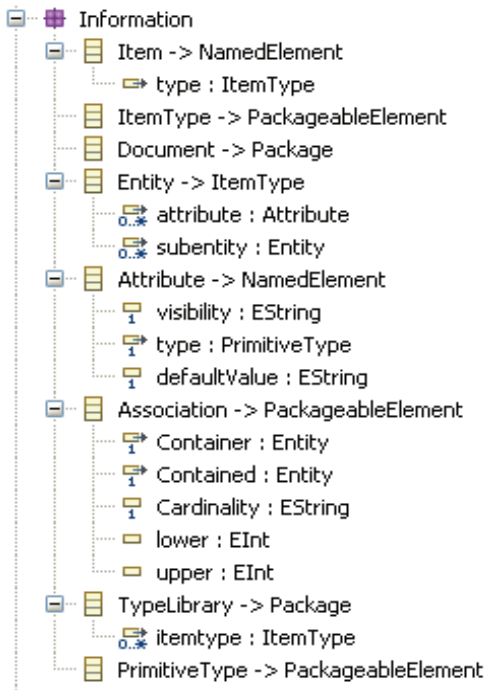

Figure A.3: The information aspect of PIM4SOA in an ecore tree editor

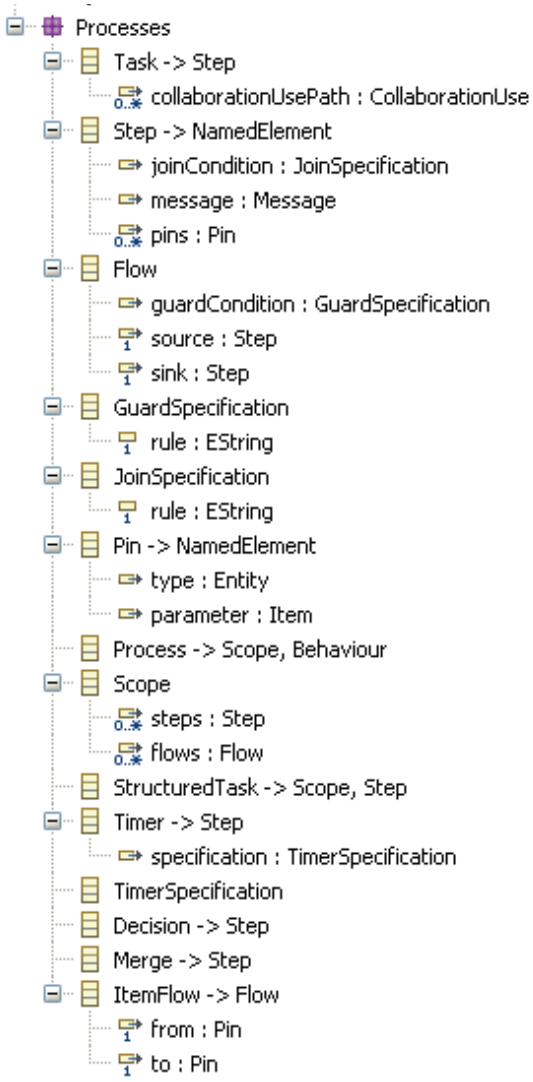

Figure A.4: The process aspect of PIM4SOA in an ecore tree editor

Bibliography

[1] Object Management Group (OMG) homepage, April 2007. http://www.omg.org.

[2] Markus Völter and Thomas Stahl. *Model-Driven Software Development : Technology, Engineering, Management*. John Wiley & Sons, June 2006.

[3] OMG. Model Driven Architecture(MDA) homepage, April 2007. http://www.omg.org/mda.

[4] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, and Tony Newling. *Service-Oriented Architecture : A Planning and Implementation Guide for Business and Technology*. John Wiley & Sons, April 2006.

[5] W3C. Web Services Architecture, 2004. http://w3.org/TR/ws-arch.

[6] Ketil Stølen and Ida Solheim. Teknologiforskning - hva er det? Technical report, STF90 A06035, Sintef ICT, 2006.

[7] Ian Sommerville. *Software Engineering (7th Edition) (International Computer Science Series)*. Addison Wesley, May 2004.

[8] Stephen J. Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, Boston, 2004.

[9] Neil Pilone, Dan with Pitman. *UML 2.0 (In a Nutshell)*. O'Reilly, June 2005.

[10] SUN Microsystems INC. Java Platform, Enterprise Edition, June 2007. http://java.sun.com/javaee/.

[11] Microsoft Corporation. .NET Framework Developer Center, June 2007. http://msdn.microsoft.com/netframework/.

[12] Watson Andy. OMG Modeling Specifications. In *JAOO'06: OMG Information Day on Business Process Management based on SOA and MDA*, April 2006.

[13] Peter Swithinbank, Mandy Chessell, Tracy Dr. Gardner, Cathrine Griffin, Jessica Man, Helen Wylie, and Larry Yusuf. *Patterns: Model-Driven Development Using IBM Rational Software Architect*. IBM RedBooks, December 2005.

[14] Andy Evans. Domain Specific Languages and MDA. Technical report, Xactium Limited, 2006.

[15] International Business Machines (IBM). Rationale Software homepage, 2006. http://www.ibm.com/software/rational.

[16] Eclipse Foundation. Model Development Tools (MDT), June 2007. http://www.eclipse.org/modeling/mdt/.

[17] Tony Clark, Andy Evans, Paul Smmut, and James Willans. *Applied Metamodelling: A Foundation for Language Driven Development*. Xactium, 2004.

[18] OASIS. Reference Model for Service Oriented Architecture, Committee Draft 1.0, February 2006. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf.

[19] Zimmerman O., Krogdahl P., and Clive G. Elements of service-oriented analysis and design. *IBM research*, 2004.

[20] Eric A. Marks and Michael Bell. *Patterns: Service-Oriented Architecture and Web Services*. IBM RedBooks, April 2004.

[21] Sprott D. and Wilkes L. Understanding service oriented architecture. *MSDN Library*, 2004.

[22] Aidima homepage, May 2007. http://www.aidima.es.

[23] Interoperability for Enterprise Software and Applications Conference I-ESA'06 . https://i-esa.laps.u-bordeaux1.fr.

[24] Ruben Costa, Oscar Garcia, Maria Jose Nunez, Pedro Malo, and Ricardo Goncalves. e-Proc TO BE Scenario for Business Interoperability. In *I-ESA'06: Enterprise Interoperability*, pages 531–540. Springer, 2006.

[25] OMG. UML Profile and Metamodel for Services (UPMS), Request for Proposal, November 2006. http://www.omg.org.

[26] Cambridge Advanced Learner's Dictionary, June 2007. http://dictionary.cambridge.org.

[27] Merriam-Webster's Online Dictionary, June 2007. http://www.m-w.com.

[28] Simon Johnston. Modeling service-oriented solutions. Technical report, IBM, 2005.

[29] Simon Johnston. UML 2.0 Profile for Software Services. Technical report, IBM, 2005.

[30] Philippe Kruchten. Introduction to the Rational Unified Process. *ICSE*, 00:703, 2002.

[31] Ali Arsanjani and Abdul Allam. Service-Oriented Modeling and Architecture for Realization of an SOA. *SCC*, 0:521, 2006.

[32] OMG. UML Profile for Enterprise Distributed Object Computing Specifications, April 2007. http://www.omg.org/technology/documents/formal/edoc.htm.

[33] Enterprise Collaboration Architecture (ECA), v1.0, formal/2004-02-01, April 2007. http://www.omg.org/cgi-bin/doc?formal/2004-02-01.

[34] Platform Independent Model for Service Oriented Architecture. http://sourceforge.net/projects/pim4soa.

[35] ATHENA IP. project homepage, April 2007. http://www.athena-ip.org.

[36] Gorka Benguria, Xabier Larrucea, Brian Elvesæter, Tor Neple, Anthony Beardsmore, and Michael Friess. A Platform Independent Model for Service Oriented Architectures. In *I-ESA'06: Enterprise Interoperability New Challenges and Approaches*, pages 23–35. Springer, 2006.

[37] OMG. Business Process Definition Metamodel(BPDM), Request for Proposal, March 2003. http://www.omg.org/cgi-bin/doc?bei/03-01-06.

[38] Fischer Klaus, Elvesæter Brian, Berre Arne-Jørgen, Hahn Christian, Madrigal-Mora Cristian, and Zinnikus Ingo. Model-Driven Design of Interoperable Agents. In *Second international I-ESA'06: Interoperability for Enterprise Software and Applications*, pages 119–131, 2006.

[39] Microsoft Visual Studio 2005: Domain-Specific Language Tools. http://msdn.microsoft.com/vstudio/DSLTools.

[40] Jack Greenfield and Keith Short. Software Factories. Assembling applications with patterns, models, frameworks and tools. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2003.

[41] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, August 2004.

[42] Erik Braathen. A practical evaluation of Software Factories and the Microsoft Domain Specific Language approach for developing Web Services. Master's thesis, Universitet i Oslo (UIO), 2005.

[43] Eclipse Foundation. Graphical Modeling Framework (GMF), May 2007. http://www.eclipse.org/gmf.

[44] Eclipse Foundation. Eclipse homepage, May 2007. http://www.eclipse.org.

[45] Eclipse Foundation. Eclipse Modeling Framework (EMF), May 2007. http://www.eclipse.org/emf.

[46] Eclipse Foundation. Graphical Editing Framework (GEF), May 2007. http://www.eclipse.org/gef.

[47] SUN Microsystems INC. Java technologies, May 2007. http://www.java.sun.com.

[48] Bill Moore, David Dean, Anna Gerber, Gunnar Wagenknecht, and Philippe Vanderheyden. *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. Redbooks, 2004.

[49] Eclipse Foundation. Display a UML Diagram using Draw2D, May 2007. http://www.eclipse.org/articles/Article-GEF-Draw2d/GEF-Draw2d.html.

[50] Frederic Plante. Introducing the GMF Runtime, January 2006. www.eclipse.org/articles/Article-Introducing-GMF/article.html.

[51] MetaCase homepage, May 2007. http://www.metacase.com.

[52] MetaCase. Abc to metacase technology. Technical report, MetaCase, 2004.

[53] Juha pekka Tolvanen. Metaedit+: Integrated modeling and metamodeling environment for domain-specific languages. In *Inproceedings of OOPSLA'06*, pages 690–691. ACM, 2006.

[54] MetaCase. Metaedit+ version 4.5: User's guide, 2006. http://www.metacase.com.

[55] Xactium. Xactium limited. http://www.xatium.com.

[56] ATHENA A6. Specification of a basic architecture reference model. D.A6.1, 2005. (downloadable from Athena IP project site).

[57] Eclipse Foundation. The Standard Widget Toolkit(SWT) homepage, May 2007. http://www.eclipse.org/swt.

[58] Steve Burback. How to use Model-View-Controller (MVC). *The UIUC Smalltalk Archive*, 1992. University of Illinois at Urbana-Champaign.

[59] Eclipse Foundation. GMF Developer Guide. http://help.eclipse.org, May 2007.

[60] Eclipse Foundation. Eclipse Modeling Framework Technologies (EMFT), Juni 2007. http://www.eclipse.org/emft.

[61] Eclipse Foundation. Java Emitter Templates(JET) introduction, May 2007. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html.

[62] OASIS. Unified Business Language (UBL) v.2.0, December 2006. http://docs.oasis-open.org/ubl/os-UBL-2.0/UBL-2.0.pdf.

[63] OMG. Business Process Modeling Notations(BPMN), February 2006. http://www.bpmn.org.

[64] OMG. *UML Profile and Metamodel for Services - for Heterogeneous Architectures (UPMS-HA), Initial submission*, June 2007. http://www.omg.org/cgi-bin/doc?ad/07-06-02.

[65] Eclipse Foundation. GMF Tutorial Part 4, May 2007. http://wiki.eclipse.org/index.php/GMFTutorial_Part_4.

[66] Andreas Limyr. Graphical editor for UML 2.0 Sequence Diagrams. Master's thesis, UIO, 2005.

Acronyms and Abbreviations

ATHENA IP Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications Integrated Project

BPDM Business Process Definition Metamodel

BPEL Business Process Execution Language

BPM Business Process Modeling

BPMN Business Process Modeling Notation

CASE Computer Aided Software Engineering

CBD Component Based Development

CCA Component Collaboration Architecture

CIM Computional Independent Model

DSL Domain Specific Language

EA Enterprise Architecture

ECA Enterprise Component Architecture

EDOC Enterprise Distributed Object Computing

EMF Eclipse Modeling Framework

EMFT Eclipse Modeling Framework Technologies

EMOF Essential Meta Object Facility

GEF Graphical Eclipse Framework

GMF Graphical Modeling Framework

IDE Integrated Development Environment

JDT Java Development Tools

JET Java Emitter Templates

LDD Language Driven Development

MDA Model Driven Architecture

MDD Model Driven Development

MDF Module Description File

MDSD Model Driven Software Development

MED4SOA Metamodel-based Editor for Service Oriented Architecture

MOF Meta Object Facility

MVC Model-View-Controller

OASIS Organization for the Advancement of Structured Information Standards

OCL Object Constraint Language

OMG Object Management Group

OOAD Object Oriented Analysis and Design

P2P Peer to Peer

PIM Platform Independent Model

PIM4SOA Platform Independent Model for Service Oriented Architecture

PDE Plug-in Development Environment

PSM Platform Specific Model

QoS Quality of Service

RFP Request for Proposal

RFQ Request for Quotation

RSM Rational Software Modeler

RUP Rational Unified Process

SMEs Small and Medium Enterprises

SOA Service Oriented Architecture

SOMA Service Oriented Modeling and Architecture

SWT Standard Widget Toolkit

UBL Unified Business Language

UML Unified Modeling Language

UPMS UML Profile and Metamodel for Service

WSDL Web Service Description Language

W3C World Wide Web Consortium

XMI XML Metadata Interchange

XML eXtensible Markup Language